

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**AN INCREMENTAL GENETIC ALGORITHM AND
NEURAL NETWORK FOR CLASSIFICATION AND
SENSITIVITY ANALYSIS OF THEIR
PARAMETERS**

by

Gözde BAKIRLI

September, 2009

İZMİR

AN INCREMENTAL GENETIC ALGORITHM AND NEURAL NETWORK FOR CLASSIFICATION AND SENSITIVITY ANALYSIS OF THEIR PARAMETERS

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Master of Science in
Computer Engineering**

**by
Gözde BAKIRLI**

**September, 2009
İZMİR**

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**AN INCREMENTAL GENETIC ALGORITHM AND NEURAL NETWORK FOR CLASSIFICATION AND SENSITIVITY ANALYSIS OF THEIR PARAMETERS**” completed by **GÖZDE BAKIRLI** under supervision of **PROF. DR. ALP KUT** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Alp Kut

Supervisor

Yrd. Doc. Dr. Derya Birant

(Jury Member)

Yrd. Doc. Dr. Reyat Yılmaz

(Jury Member)

Prof.Dr. Cahit HELVACI

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank to my supervisor, Prof. Dr. Alp Kut, for his support, supervision and useful suggestions throughout this study. I am also highly thankful to Dr. Derya Birant for her valuable suggestions throughout study.

I owe my deepest gratitude to my family. This thesis would not have been possible unless their unflagging love and support. I am indebted to my father Mustafa Ali Bakırlı, my mother Beyza Bakırlı and my brother Demir Bakırlı for their care and love.

Special thanks are directed to TUBITAK for their financial support throughout two years. Thanks to TUBITAK BİDEB, I could attain necessary hardware and software equipments and I could allocate more time to my thesis.

Gözde BAKIRLI

AN INCREMENTAL GENETIC ALGORITHM AND NEURAL NETWORK FOR CLASSIFICATION AND SENSITIVITY ANALYSIS OF THEIR PARAMETERS

ABSTRACT

This study proposes classification by using algorithms that are inspired by computation in biological systems and to compare of them. These are genetic algorithm and neural network. New incremental genetic algorithm and new incremental neural network algorithm are developed for classification for efficiently handling new transactions. To achieve incremental classification, a specific model that includes all information about a train operation, rules for each class for genetic algorithm and weight values for neural network is created after each training operation. Later, these models are used for testing, correctness test, comparing models and incremental classification. With that new incremental method, training time gets smaller for new dataset. Experimental results proof that assumption. This paper introduces that new method and importance of that method. This study also includes the sensitivity analysis of the incremental and traditional genetic algorithm parameters and neural network parameters. In this analysis, many specific models were created using the same training dataset but with different parameter values, and then the performances of the models were compared. To achieve these operations two tools are developed for both genetic algorithm and neural network and all of these investigations are done by using these tools.

Keywords: Genetic Algorithm, Neural Network, Classification, Data Mining, Incremental Mining, Sensitivity Analysis

ARTIMLI GENETİK ALGORİTMA VE YAPAY SİNİR AĞLARI İLE SINIFLAMA VE PARAMETRELERİNİN HASSASİYET ANALİZİ

ÖZ

Bu çalışmanın amacı biyolojik sistemden etkilenilip geliştirilen genetik algoritma ve yapay sinir ağları ile sınıflama yapmak ve artımlı algoritmalarını geliştirip geleneksel yöntem ile karşılaştırmasını yapmaktır. Bu amaç doğrultusunda her eğitim aşamasından sonra o eğitime özgü bir model oluşturulmaktadır. Bu model, eğitim ile ilgili tüm bilgileri ve eğitim aşamasından sonra elde edilen çıktıları saklamakta ve bu bilgiler daha sonra test işlemi, modelin doğruluk testi, modellerin performans hesaplamaları, performanslarının karşılaştırılması ve artımlı sınıflama sağlamak için kullanılmaktadır. Artımlı sınıflama geleneksel yöntemle göre daha kısa eğitim zamanı ile daha fazla performans sağlamıştır. Bu amaca yönelik yapılmış deneysel gözlemler bu performans kazancını ispatlamaktadır. Ayrıca genetik algoritma ve yapay sinir ağları parametrelerinin değerleri değiştirilip, eğitim işlemlerinin sonuçları karşılaştırılarak hassasiyet analizi yapılmıştır. Tüm bu işlemleri gerçekleştirmek amacıyla genetik algoritma ve yapay sinir ağları için ayrı iki sınıflama aracı geliştirilmiştir.

Anahtar Sözcükler: Genetik Algoritma, Yapay Sinir Ağları, Sınıflama, Veri Madenciliği, Artımlı Sınıflama, Hassasiyet Analizi

CONTENTS

Page

M.Sc THESIS EXAMINATION RESULT FORM.....	ii
ACKNOWLEDGMENTS	iii
ABSTRACT.....	iv
ÖZ	v
 CHAPTER ONE - INTRODUCTION	 1
1.1 Data Mining	1
1.2 Classification.....	1
 CHAPTER TWO – GENETIC ALGORITHM.....	 3
2.1 Related Works.....	3
2.2 Genetic Algorithm for Classification.....	5
2.3 Fitness Function	7
2.4 Parent Selection Techniques	10
2.4.1 Roulette-Wheel Selection.....	10
2.4.2 Tournament Selection	10
2.4.3 Top Percent Selection	10
2.4.4 Best Selection.....	11
2.4.5 Rank Selection	11
2.4.6 Random Selection	11
2.5 Crossover	12
2.5.1 One Point Crossover	12
2.5.2 Two-Point Crossover	13
2.5.3 Uniform Crossover.....	13
2.6 Mutation	15
2.7 Termination Criteria.....	15

2.7.1	Generation Number	16
2.7.2	Fitness Value	16
2.8	Experimental Results	22
2.8.1	Description of the Dataset	22
2.8.2	Determining Crossover and Mutation Probability	24
2.8.3	Importance of Elitism.....	26
2.8.4	Population Size	27
2.8.5	Traditional vs. Incremental GA	29
2.8.6	Classification Accuracy	32
2.9	Interface.....	34
2.9.1	Training	35
2.9.2	Comparing Models.....	39
2.9.3	Testing.....	41
2.9.4	Incremental GA.....	44
CHAPTER THREE – NEURAL NETWORK.....		49
3.1	Related Works	49
3.2	Neural Network.....	49
3.2.1	Simple Single Unit Network	49
3.2.2	Activation Functions	51
3.2.3	Termination Criteria.....	53
3.2.4	Neural Network Types	54
3.3	Approach	60
3.3.1	Incremental Approach.....	60
3.3.2	Performance Calculation of the Models.....	62
3.4	Experimental Results	62
3.4.1	Model Construction.....	62
3.4.2	Description of the Dataset.....	64
3.4.3	Iteration Number	64
3.4.4	Hidden Layers	66

3.4.5	Learning Rate (μ)	67
3.4.6	Neural Network Types	68
3.4.7	Incremental vs. Traditional Backpropagation NN	69
3.5	Interface.....	71
3.5.1	Backpropagation	71
3.5.2	SLP	73
3.5.3	MLP	74
3.5.4	SOM	75
3.5.5	Testing.....	77
3.5.6	Comparing Models.....	79
3.5.7	Incremental NN.....	80
CHAPTER FOUR - CONCLUSION		83
4.1	Conclusion for GA	83
4.2	Conclusions for NN	84
4.3	GA vs. NN for Incremental Classification.....	85
REFERENCES.....		86

CHAPTER ONE

INTRODUCTION

1.1 Data Mining

Data Mining is the process of extracting hidden patterns from large datasets. Data mining has been widely used in many areas, such as marketing, banking and finance, medicine and manufacturing. There are many data in these areas. Data mining is the most widely used method to process these data. There are commonly four tasks of data mining. These are; Classification, Clustering, Regression and Association rule learning.

1.2 Classification

Classification is a procedure in which individual items are placed into groups based on quantitative information on some characteristics inherent in the items. In the classification process, a collection of labelled classes is provided and a training set is used to learn the descriptions of classes. Classification rules are discovered and then these rules are used to determine the most likely label of a new pattern. The most widely used classification techniques are neural networks, decision trees, k-nearest neighbours, support vector machines, and naive Bayes.

Neural Network is one of most widely used classifiers. Much successful classification can be done with Neural Network. There are so many examples for Neural Network classification. But many of them don't support testing operations, incremental NN, performance calculation of models and models comparison.

Genetic Algorithm is not accepted between most widely used classifiers as a classifier, but so successfully classification can be done with Genetic Algorithm. There have been

a few samples for genetic algorithm when it is used as a classifier. None of them supports testing operations, models comparing and incremental GA.

The aim of that study is to show importance of saving information of each train operation. A model is created for each training. This model includes all inputs and outputs of train operation. These information are used for testing (classes finding of new patterns), comparisons of models, correctness testing and decrementing training time for dataset which is updated regularly.

This study also presents the sensitivity analysis of the GA parameters such as crossover probability, mutation probability, with/without elitism and population size and NN parameters such as input and output neuron numbers, hidden layer numbers, activation function.... The aim of this analysis is to evaluate the performances of the classification models which are constructed using the same training dataset with different GA parameter values and different NN parameter values. Each classification model for GA (classifier) consists of input parameters (crossover and mutation probabilities, population size etc.), applied techniques (parent selection type, crossover type, different termination criteria etc.), and outputs (average fitness value, classification rules etc.) related to training process. And each model for NN consists of input parameters (NN type, Network Layer number, network dimensions etc.) and weight values. The models are compared by applying n-fold cross validation method. In order to implement all these experiments, two tools are developed, named Generic Genetic Classifier Tool and Neural Network Modeller.

CHAPTER TWO

GENETIC ALGORITHM

2.1 Related Works

Genetic Algorithms are a family of computational models motivated by the process of natural selection in biological system. Evolutionary computing concept is appeared in the 1960's by I.Rechenberg. GA was first developed by Holland in 1975 and then improved by other many researchers (Booker, Goldberg & Holland, 1989). Currently, GA is one of the most important techniques of artificial intelligence. GAs are used for soft constraint satisfaction, scheduling problems, finding game strategies, and so forth.

The basis of genetic algorithm is "natural selection". That means, individuals who have sufficient features to live, are transferred in next generation, and other individuals who are not good enough, disappear. The stronger candidates remain in the population, the weaker ones are discarded (Shapiro, 2001). So new generation gets closer to the best solution at each step and this operation goes on until termination criteria are met. For the basic concept of genetic algorithms, please refer to Goldberg (1989).

In recent years, only in a few studies, GAs has been applied for classification problem to discover classification rules. Ishibuchi, Nakashima, and Murata (2001) constructed a fuzzy classifier system in which a population for fuzzy if-then rules is evolved from genetic algorithms. Avcı (2009) implemented classification method by combining genetic algorithm and support vector machine techniques. Fan, Chen, Ma and Zhu (2007) created an approach for proposal grouping, in which knowledge rules are designed to interact with proposal classification, and the genetic algorithm is developed to search for the expected groupings. Yuen et al. (2009) proposed a hybrid model which combines genetic algorithm and neural network for classifying garment defects. Kwong,

Chang and Tsim (2008) used genetic algorithm to discover knowledge about the fluid dispensing. Dehuri, Patnaik, Ghosh and Mal (2008) used an elitist multi-objective genetic algorithm for mining classification rules from large databases. Yılmaz, Yıldırım, and Yazıcı (2007) used genetic algorithm to make classification segments of video to objects.

According to the review of GA-based classification methods, previous studies use either traditional genetic algorithm or combination of genetic algorithm with another AI technique such as fuzzy, neural network. They don't propose the incremental usage of the genetic algorithm for classification when new data is added to the existing dataset.

The problem of incrementally updating mined patterns on changes of the database, however, has been proposed for other data mining tasks such as clustering, association rule mining. Lin, Hong, Lu (2009) propose an efficient method for incrementally modifying a set of association rules when new transactions have been inserted to the database. Lühr and Lazarescu (2009) introduce an incremental graph-based clustering algorithm to both incrementally cluster new data and to selectively retain important cluster information within a knowledge repository. Fan, Tseng, Chern, and Huang (2009) propose an incremental technique to solve the issue of added-in data without re-implementing the original rough set based rule induction algorithm for a dynamic database.

Sensitivity analysis is the study to determine how a given model output depends upon the input parameters. (Saltelli, 2008) In other words, it is the process of varying input parameters over a reasonable range and observing the relative change in model response. It is an important process for checking the quality of a given model, as well as a powerful tool for checking the robustness and reliability of the model. A sensitivity analysis can be conducted by changing each parameter value by $\pm 10\%$ and $\pm 50\%$ (Cacuci, 2003). This study compares the performance of the classification models constructed by different GA parameter settings.

2.2 Genetic Algorithm for Classification

Each phase in GA (Figure 2.1) produces a new generation of potential solutions for a given problem. In the first stage, an initial *population*, which is a set of encoded bit-strings (*chromosomes*), is created to initiate the search process. The performance of the strings is then evaluated with respect to the *fitness function* which represents the constraints of the problem. After the sorting operation, the individuals with better performance (fitness value) are selected for a subsequent genetic manipulation process. The selection policy is responsible for assuring survival of the best-fit individuals. In the next stages, a new population is generated using two genetic operations: *crossover operation* (recombination of the bits/genes of each two selected strings/chromosomes) and *mutation operation* (alteration of the bits/genes at one or more randomly selected positions of the strings/chromosomes). This process is repeated until certain criteria are met.

1. Initial population of n chromosomes is created randomly
2. Fitness value of each chromosome is calculated by using fitness $f(x)$ function
3. Chromosomes are sorted in descending order according to their fitness values
4. Parents are selected by using selection techniques
5. Crossover operation is made according to crossover probability
6. Mutation is applied to chromosomes after crossover according to mutation probability
7. Old generation is replaced with new generation
8. Repeat these steps until termination criteria is met.

Figure 2.1 Basic genetic algorithm

Search Space is a subset which includes all possible solutions of the problem.

Population is a subset of n randomly chosen solutions from the search space. For data mining, population is created randomly class number times. For example; if there are five classes in dataset then population is created five times. Because train operation for each class runs for five times. For example there are two classes, 'YES' and 'NO', so

firstly, population is created for ‘YES’ class, and finds solutions, rules for that class, and then the same operation is made for ‘NO’ class.

Population consists of *chromosomes*. Chromosomes are strings which are possible solutions of that problem. Length of chromosomes is determined while population is being created randomly at the beginning, I look the number of the difference values of each attribute, and then I create chromosomes by looking these numbers. To understand clearly look at nursery dataset, which is one of datasets I study.

In nursery dataset there are nine attributes. The last one is class value. After number of attributes is determined, dataset is searched for nine times. In every search we find how many different values for relevant attribute, there are. For example for the first attribute which name is ‘parent’, there are three different values, which are ‘usual’, ‘pretentious’ and ‘great_preat’. So the length of the part of chromosome for that attribute is three. For the first value, ‘usual’, our string part is ‘100’, for the second value, ‘pretentious’, our string part is ‘010’, and for the last value, ‘great_preat’, our string part is ‘001’. So the first part of the chromosomes can be in the following forms;

100: if parent=usual

010: if parent=pretentious

001: if parent=great_preat

110: if parent=usual or parent=pretentious

011: if parent=pretentious or parent=great_preat

101: if parent=usual or parent=great_preat

111: if parent=usual or parent=pretentious or parent=great_preat (in that situation that attribute is noneffective for relevant class.)

At the beginning population is created randomly, that means these string parts are created randomly. For example while a chromosome is being created, for ‘parent’

attribute, the part of that chromosome will be one of above string parts. And this operation is done for every attributes.

For each attribute we create these string parts, and then we piece together these parts. This operation is made for population size times.

After population creation, all individuals' fitness values are calculated.

2.3 Fitness Function

This function is used for determining how acceptable individual is. There is not a fixed function to calculate fitness value. Fitness function depends on problem. But for classification with rule discovery by using genetic algorithm there is only one fitness function Freitas, AA(1999).

Rule for each class is of the form “IF condition THEN class”. Fitness value of that rule is predictive accuracy of that. Predictive accuracy is found by computing confidence factor of that rule.

Definition 1. Confidence factor of rule

$$\text{Confidence factor} = \frac{\#(\text{condition \& class})}{\#(\text{condition})}$$

$\#(\text{condition})$: the number of examples in that *condition*

$\#(\text{condition \& class})$: the number of examples in that *condition* and have that *class*. That is, class number predicted by that rule.

For example there is 10 rules which are same with '*condition*' (# (*condition*) = 10) and 3 of them are from *class* (# (*condition* & *class*) = 3), then *confidence factor* of that rule is 3/10.

But there is a problem. For example # (*condition*) = 1 and # (*condition* & *class*) = 1, that is there is only one pattern which is "IF *condition* THEN *class*", but according *confidence factor* formula, result is "1/1 = 100%". But it should not be. Because maybe that rule doesn't define of class "C".

To overcome with that problem we use another component.

TP: True Positives: Number of examples satisfying *condition* and *class*

FP: False Positives: Number of examples satisfying *condition* but not *class*

FN: False Negatives: Number of examples not satisfying *condition* but satisfying *class*

TN: True Negatives: Number of examples not satisfying *condition* nor *class*

In that situation *confidence factor* is equals to;

Definition 2. Formula of confidence factor of rule is as follows;

$$\text{Confidence factor} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

We know that *confidence factor* is not enough for calculating fitness value of the rule. So another component is needed for fitness value. It is "completeness". Completeness is used for determining how complete the rule is.

Definition 3. Formula for completeness which shows how complete current rule is

$$\text{Comp} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

So now the fitness function such as:

Definition 4. Fitness value for classification operation is calculated as follows;

$$\text{Fitness} = \text{confidence factor} \times \text{Comp}$$

This is the fitness function that is used in that project for classification.

Training operation is applied for each class, and then average fitness value of each operation is calculated. This is the result fitness value, and this value effects performance of the model.

Definition 5. Population fitness is defined as:

$$\text{Population_Fitness} = \sum_{i=1}^{|\text{chr}|} \text{Fitness}(\text{chr}_i)$$

where $|\text{chr}|$ is the number of chromosome in the population (population size), chr_i is the one chromosome in the population.

Definition 6. Result fitness value calculation is shown below:

$$\text{result_fitness} = \left(\sum_{i=1}^{|\text{c}|} \text{Population_Fitness}(p_i) \right) / |\text{c}|$$

where $|\text{c}|$ is the total class number, p_i is the population for i^{th} class.

Fitness value is calculated for every individual at the population. After this operation individuals are sorted according to their fitness values.

After sorting, parent selection is made according to selection techniques for cross-over operation.

2.4 Parent Selection Techniques

2.4.1 Roulette-Wheel Selection

This is the most widely used selection technique. Roulette wheel selection is implemented as follows

1. Find total fitness of the population.
2. Generate n randomly, between 0 and total fitness
3. Return the first population member whose fitness added to the preceding population members is greater than or equal to n .

2.4.2 Tournament Selection

As its name, there is a tournament among individuals. A few individuals are selected from the population randomly. The individual which has the highest fitness value is selected between these individuals.

2.4.3 Top Percent Selection

In that selection technique, top n percent of population is used. For example there are 100 individuals in the population and n is 20, then we select randomly one individual between first 20 individuals.

2.4.4 *Best Selection*

Parents are first two individuals of the population which has highest fitness value. The purpose of that method is to accelerate training by avoiding individuals which have poor fitness value. This method can not work when best individuals are not good enough. Because other individuals which have not high fitness value, may have successful performance and may get close solution after crossover or mutation operations.

2.4.5 *Rank Selection*

Tournament selection will have problems when the fitnesses differ very much. For example, if the best chromosome fitness is 90% of all the roulette wheel then the other chromosomes will have very few chances to be selected.

Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness *1*, second worst *2* etc. and the best will have fitness *N* (number of chromosomes in population).

2.4.6 *Random Selection*

Parents are selected randomly with that method. A number 'n' is generated between "1" and population size. After that operation nth member of population is selected as parent.

There are two alternatives for cross-over. These are not kind of cross-over techniques, these are choices to get better performance and to decrement time which we need to reach to solution. These are *Steady-State Selection* and *Elitism*. The mission of steady-state selection is transferring individuals which have high fitness value, into next generation without any operation. The purpose of that, to save good individuals and not

losing them. The purpose of Elitism is similar, but only first two individuals which have highest fitness value are added into next generation directly with elitism.

After parents are selected, cross-over operation is applied.

2.5 Crossover

In that operation genetic information of two individuals are merged. The purpose of that operation is to find best individual. There are three main cross-over techniques for data mining.

2.5.1 One Point Crossover

Each parent divides into two parts according to a fix number which is between 1 and string length or a number which is generated randomly. And the first part of first parent and the second part of the second parent are merged and first child is created. The second part of first parent and the first part of second parent create second child.

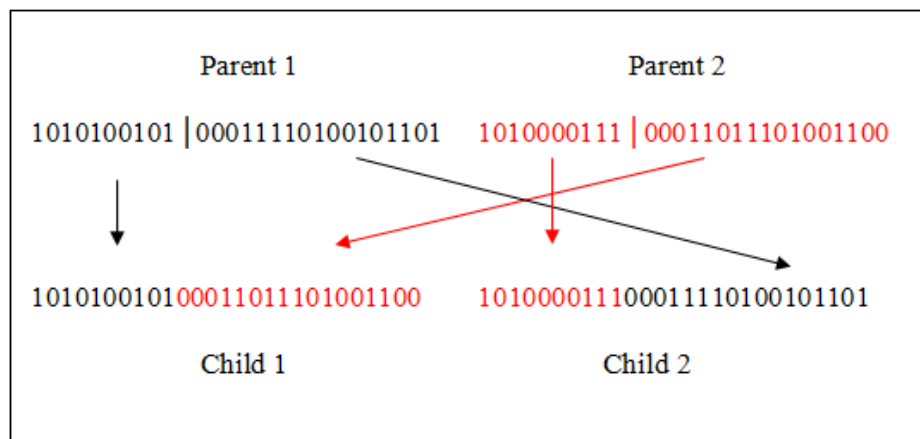


Figure 2.2 One point crossover

2.5.2 Two-Point Crossover

Each parent divides into three parts according to fix two numbers which are between 1 and string length or which are generated randomly. The first and third parts of the first parent and the second part of the second parent create first child. The second part of the first parent and the first and third parts of the second parent create second child.

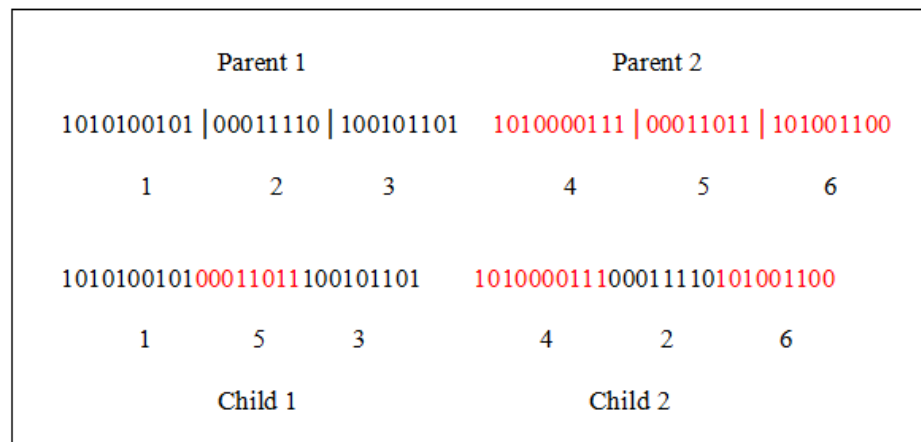


Figure 2.3 Two point crossover

2.5.3 Uniform Crossover

A string is generated randomly. The mask determines which bits are copied from first parent and which from the other parent. The bits which are “1” in mask, show that these bits will be copied from the first parent, and “0” bits are for second parents.

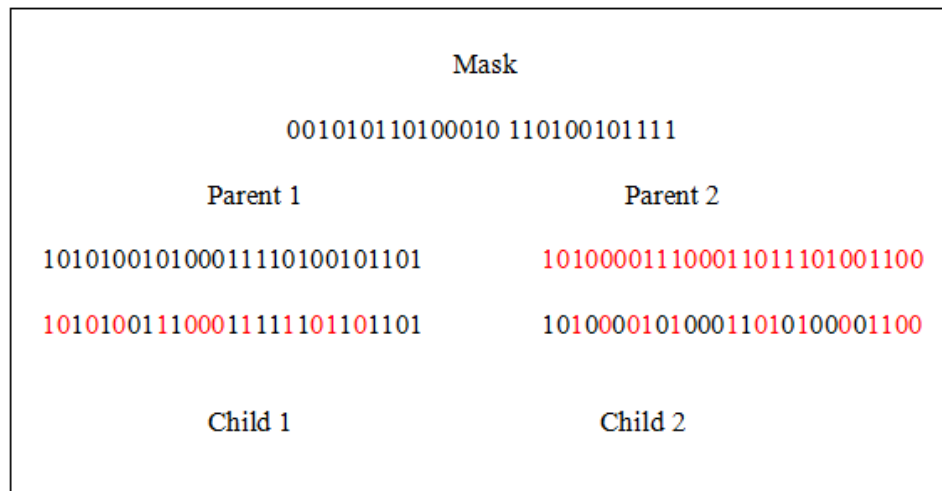


Figure 2.4 Uniform crossover

Crossover operation is shown in Figure 2.5 as a pseudo code.

```

Function cross_over (parent1, parent2)
1. Generate a 'n' number between 0 and 1,
   if n<cross_over_probability then go to 2, else return;
2. if cross_over is one_point then , generate ' x' number between 1 and
   chromosome_length
   child1=parent1(0,x)+parent2(x,length) (as string)
   child2=parent2(0,x)+parent1(x,length) (as string)
   End if
3 if cross_over is two_point then, generate 'x1' and 'x2' numbers between 1 and
   chromosome_length
   if x2<x1 excahge them
   child1=parent1(0,x1)+parent2(x1,x2-x1)+parent1(x2,length)
   child2=parent2(0,x1)+parent1(x1,x2-x1)+parent2(x2,length)
   End if
4. if cross_over is uniform then
   For i=0 to chromosome_length
       Generate a 'm' number 0 or 1 randomly
       If m=0 then
           Add into child1 chromosome ith bit of parent2
           Add into child2 chromosome ith bit of parent1
       End if
       If m=1 then
           Add into child1 chromosome ith bit of parent1
           Add into child2 chromosome ith bit of parent2
       End if
   End for
End if
Return chil1, child2

```

Figure 2.5 Crossover

Cross-over operation is applied according to cross-over probability. This probability value is between 0 and 1. Before operation a random number is generated between 0 and 1. If this number smaller then probability, cross-over is applied, otherwise parents are transferred directly, without any operation.

2.6 Mutation

After cross-over operation, mutation is applied. Mutation is a random deformation of the strings with a certain probability. This probability is similar with cross-over probability. If random n number smaller then mutation probability, mutation operation is made. Certain bits of string are changed with that operation. The aim of mutation operation is avoiding local minimum. Figure 2.6 shows pseudo code of mutation operation.

```

Mutation (chromosome)
1. Generate a 'n' number between 0 and 1
   If n<mutation_probability go to 2, else return;
2. Generate a 'm' number between 1 and 3
   For i=0, i<m
       Generate a 'x' number between 0 and chromosome length
       Change the xth bit "0 to 1" or "1 to 0"
   End for
3. return chromosome;

```

Figure 2.6 Mutation

2.7 Termination Criteria

These cross-over and mutation operations attend until termination criteria are met. There are two most widely used termination criteria. These are generation number and fitness value.

2.7.1 *Generation Number*

As its name, there is a given number which shows how many new generations will be created. For example if generation number is 50, then after 50 generation, loop of generation is terminated regardless of average fitness of the population. Disadvantage of that method, training can be terminated before reaching solution.

2.7.2 *Fitness Value*

In that situation fitness value of population needs to reach given fitness value to terminate. Disadvantage of that method, wrong fitness value can be determined as a threshold, so when loop is terminated, there can be no solution of the problem. For example 25 is determined as a threshold fitness value. But maybe there will be a population which has 30 fitness value, and of course closer to best solution. But because of false threshold, this best solution can not be reached. On the other hand wrong fitness value causes infinite loop. For example population's best fitness value is 30, but at the beginning 35 is selected as a threshold that can never be reached. In that situation loop can not terminate.

When termination criterion has been reached, there is a population that includes rules for one class. These rules or this rule characterize of current rule. After rules are found for one class, the same operation is done for another class and rules of that class are found.

The purpose of that study is to achieve incremental GA firstly. To realize that, the incremental genetic algorithm for classification in Figure 2.7, is developed.

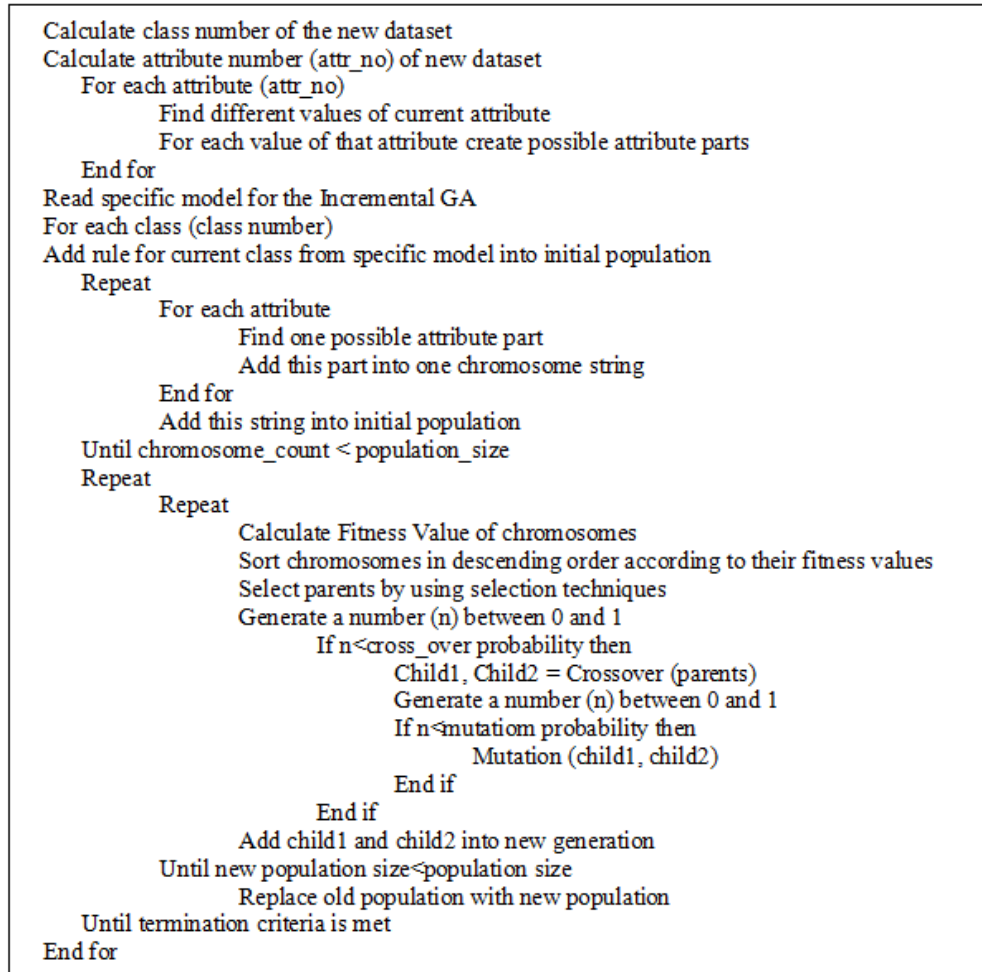


Figure 2.7 Incremental GA

In training process, the main difference between traditional GA and our incremental GA is the generation of the initial population. While, in traditional GA, initial population is generated fully random, in incremental GA, the classification rules are also added into the randomly generated population. Experimental results show that the results obtained from traditional and incremental GA are the same, but incremental GA reduces the generation number and decreases time which we need to reach to solution. The flowchart of the proposed training process is depicted in Figure 2.8.

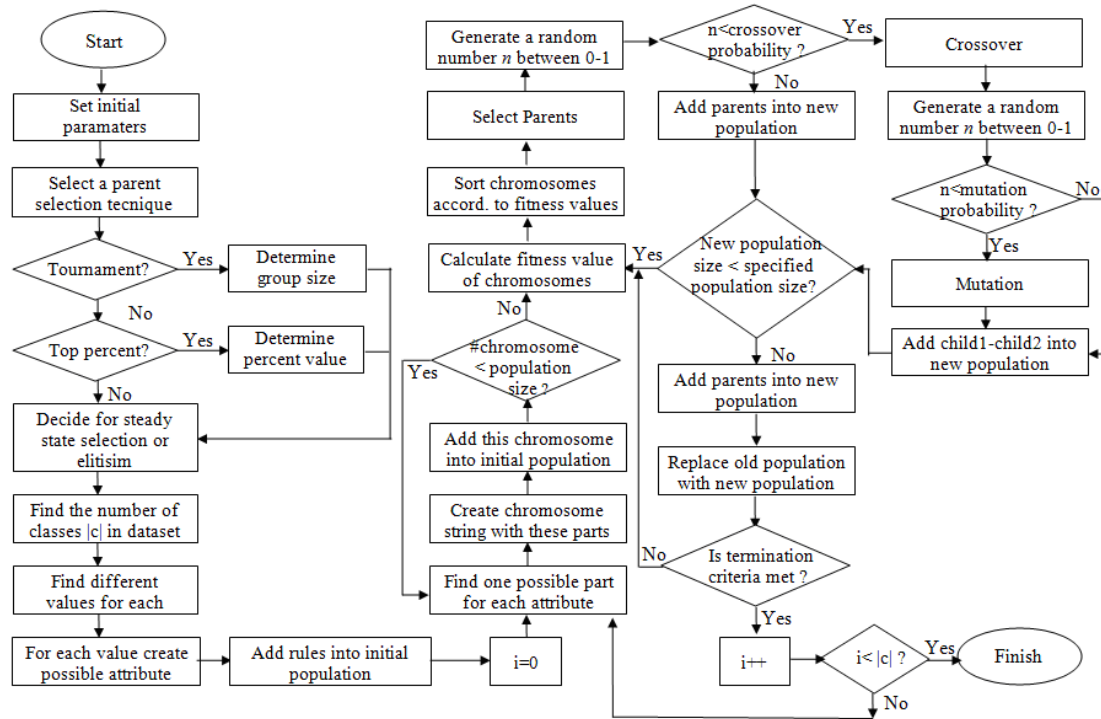


Figure 2.8 The flowchart of the proposed training process

By using above algorithm training (classification) operation is implemented as described before. After that operation there is a specific model of that operation. This model is shown in Figure 2.9.

```

Selection Type :best
Steady_State Selection :false
Elitism :true
Cross_over Probability :0,8
Mutation Probability :0,7
Population Size :100
Cross_over Type :two_point
Replace If Better :true
Replace Always :false
Termination Criteria :Generation Number
Generation Number :50
Average Fitness :35,4044097222222
Attributes :parents,has_nurs,form,children,housing,finance,social,health,class
RULES
If (parents=usual) and (has_nurs=proper) and (form=complete) and (children=1) and (housing=convenient) and (finance=co
If (parents=usual or parents=pretentious) and (has_nurs=proper or has_nurs=less_proper or has_nurs=improper) and (heal
If (health=not_recom) then class = not_recom
If (parents=usual or parents=pretentious) and (has_nurs=proper or has_nurs=less_proper) and (housing=convenient) and (
If (has_nurs=critical or has_nurs=very_crit) and (health=recommended or health=priority) then class = spec_prior

```

Figure 2.9 An example classifier model

This model includes inputs that are entered ; *Selection Type* (roulette wheel, tournament, top percent, best, rank or random) if selection type is tournament then model includes “*Group Size*” value or selection type is top percent then there will be “*Percent Value*” in the model, *Steady_State Selection* (true or false), *Elitism* (true or false), *Cross_over Probability* (between 0 and 1), *Mutation Probability* (between 0 and 1), *Population Size*, *Cross_over Type* (one_point, two_point or uniform), *Replace If Better* (true or false), *Replace Always* (true or false), *Termination Criteria* (generation number or fitness value). These are inputs. This model also includes outputs of classification operation; *Generation Number*, *Average Fitness*, *Attributes* and *Rules*.

This specific structure is used for calculation performances of models.

Performances of models depend on generation number and result fitness of the population. Performance is directly proportional to the *result_fitness* and inversely proportional to the *generation number*.

Definition 7. Performance of the model is calculated as follows, P_m stands for performance of the model, *result_fitness* is declared in **Definition 6**.

$$P_m \propto \frac{\text{result_fitness}}{\text{generation number}}$$

Result fitness and generation number effect performance. But the forcefulness of their effect is not the same. Because the main purpose of training operation with genetic algorithm, is having maximum fitness and than minimum generation number. That is fitness value has priority. Look from this perspective, training operation can not be interrupted to have small generation number. We can say that fitness should have double effect on performance calculation. In that situation the following formula is used.

Definition 8. Performance calculation when fitness value has double effect on calculation.

$$P_m = \frac{2 * \text{result_fitness}}{\text{generation number}}$$

But sometimes generation number can be more important than fitness value, or there can be a delicate balance between average fitness and generation number. So weights should be used for calculation.

Definition 9. Performance calculation with user defined weights, w1 is a weight for result fitness and w2 is a weight for generation number.

$$P_m = \frac{w1 * \text{result_fitness}}{w2 * \text{generation number}}$$

Comparisons of models' performances are done by using the pm formula in Definition 9. With these comparisons, parameters which are ideal for a dataset, can be determined.

This study provides effective way for classification for datasets which are updated regularly. To achieve that, models that are created after training operation, are used. (Figure 2.9) These specific models include rules that stand for every class in the dataset. When new patterns are added into dataset, a new classification, a new training operation is needed. Before that study all operations were applied one by one, and this causes waste of time. Because a part of that dataset is trained before and rules in that dataset are found. This study solves that problem and eliminates wasting of time. To succeed that, rules that are found before, are added into initial population. And then other operations are done. That means initial population is not created fully randomly, an intervention is applied to initial population. All classes in the new dataset are determined, and rules for

classes that were in the previous dataset, are found from the model, and added in initial populations. This provides to access expected fitness value with fewer generation number. Figure 2.10 shows that operation.

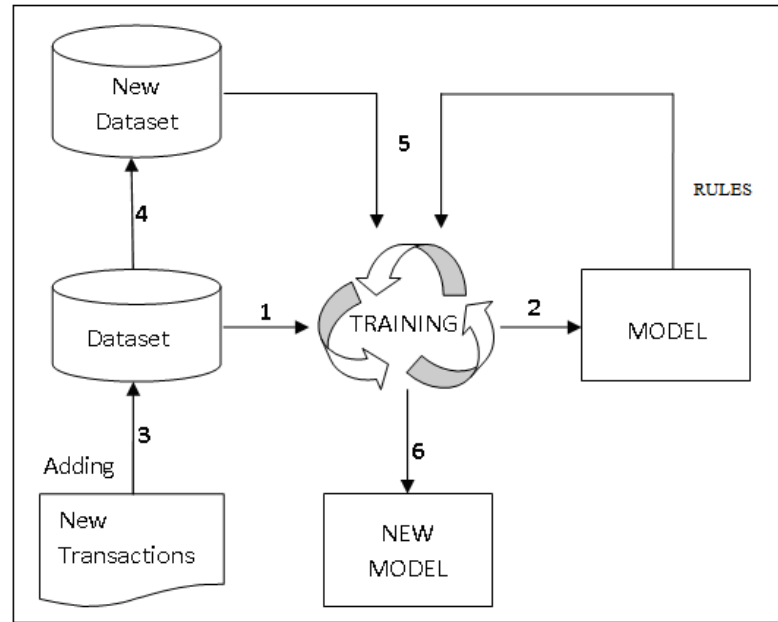


Figure 2.10 Incremental approach for handling new patterns

Initial dataset is trained and model of that dataset is created. This model includes all information and rules for that training operation. Later, new patterns are added into dataset. For example there were 15000 patterns in the initial dataset, and 1000 new patterns are added into dataset. For 1000 patterns all dataset should be trained in any case. But generation number and so training time can be reduced by using model that is created for initial dataset.

The following notations are used in the proposed model:

- S_t Selection Type
- M_p Mutation Probability
- C_p Crossover Probability

P_s	Population Size
C_t	Crossover Type
T_c	Termination Criteria
G_n	Generation Number
P_v	Percent value for top percent selection
G_s	Group size for tournament selection
GN	Generation number as a termination criteria
FV	Fitness value as a termination criteria

2.8 Experimental Results

2.8.1 *Description of the Dataset*

For the purpose of testing the performance of the proposed incremental GA, classification experiments are conducted on the real-world data "Nursery" from the UCI Machine Learning Repository (Asuncion & Newman, 2007). Nursery data consists of 12960 instances with 9 features derived from a hierarchical decision model originally developed to rank applications for nursery schools in Ljubljana, Slovenia. As shown in Table 2.1, all attributes have categorical values and the target (class) attribute has five different classes, namely, not_recom, recommend, very_recom, priority, and spec_prior.

Table 2.1 Attributes and attribute values of Nursery dataset

Attribute Name	Attribute Values
parents	usual, pretentious, great_pret
has_nurs	proper, less_proper, improper, critical, very_crit
form	complete, completed, incomplete, foster
children	1, 2, 3, more
housing	convenient, less_conv, critical
finance	convenient, inconv
social	non-prob, slightly_prob, problematic
health	recommended, priority, not_recom
class	not_recom, recommend, very_recom, priority, spec_prior

When generating initial population, chromosomes are created using binary encoding. In binary encoding, the length of a chromosome is determined by the number of the different values of each attribute. For example; if an attribute has three different values: ‘usual’, ‘pretentious’ and ‘great_preat’ then the length of the part of chromosome for that attribute becomes three as shown below.

```

100: if parent = usual
010: if parent = pretentious
001: if parent = great_preat
110: if parent = usual or parent = pretentious
011: if parent = pretentious or parent = great_pret
101: if parent = usual or parent = great_pret
111: if parent = usual or parent = pretentious or parent = great_pret
(In the last case, this attribute is ineffective for relevant class)

```

Encoding operation is done for every attributes, each attribute constitutes a string part and chromosomes are randomly constructed by the concatenation of them. The following string is one of possible chromosomes for Nursery dataset.

101-00001-1100-0110-111-01-001-100-00001

The meaning of that chromosome is;

If (parent=usual or parent=great_preat) and
 (has_nurs=very_crit) and
 (form=complete or form=comleted) and
 (children=2 or children=3) and
 (housing=convenient or housing=less_conv or housing=critical) and
 (finance=inconv) and
 (social=problematic) and
 (health=recommended)
 Then class=spec_prior

Chromosomes for initial population for one class are randomly generated in this way until the desired population size has been reached. After termination criterion is met, other initial populations are created for other classes.

2.8.2 Determining Crossover and Mutation Probability

In this section importance of crossover and mutation probabilities for fitness value are depicted. Crossover and mutation probabilities effect training time and so generation number and fitness value directly. If crossover probability is too big, needless skipping occurs, and important individuals who have big fitness value, can be lost. If crossover probability is so small then training time increases. Figure 2.11 shows importance of crossover probability and replace type. Blue line is a fitness value-crossover probability involvement when replace type is if_better, that is children are added into new generation if their fitness values are greater than their parents, otherwise parents are transferred into new generation. Pink line is for replace type is replace_always. In that situation children are added into new generation without controlling their fitness value. Table 2.2 includes parameters that are fixed for that observation. C_p s are taken between

0.3 and 0.9, 7 different values. For each C_p 10 trainings are done, and averages of these observations are taken. As it can be seen from graphic that is shown in Figure 2.11 crossover probability(C_p) effects fitness value(FV) directly. The more C_p increases, the more FV increases for fixed generation number (GN). FV measurements are listed in Table 2.3.

Mutation probability (M_p) effects FV like C_p , substantially. M_p is taken between 0.3 and 0.8. The more M_p increases, the more FV increases for fixed GN until one point. That limit is 0.7. After that point the more M_p increases, the more FV decreases. Because system starts running randomly and in some situations good individuals are lost because of oft mutation. This observation is depicted in Figure 2.12. For each M_p , 10 trainings are applied and averages of these results are taken. Fixed parameters and their values are listed in Table 2.4.

Table 2.2 Parameters of training for C_p -FV Involvement examination

S_t	Pv	Steady_State	Elitism	M_p	P_s	C_t	T_c	G_n
Toppercent	10	False	False	0.5	100	two-point	GN	50

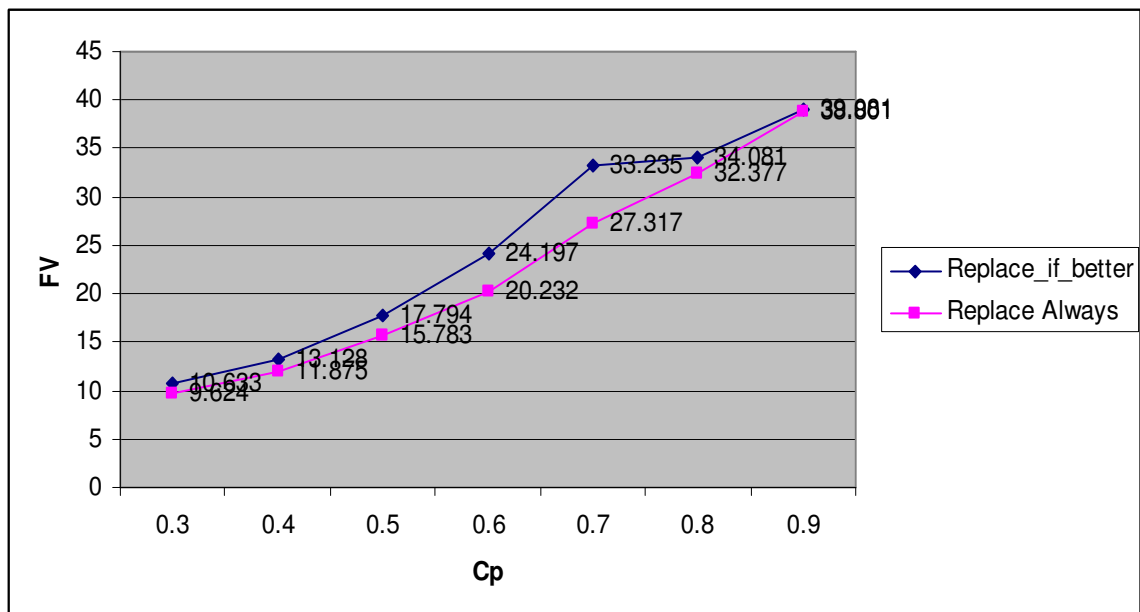
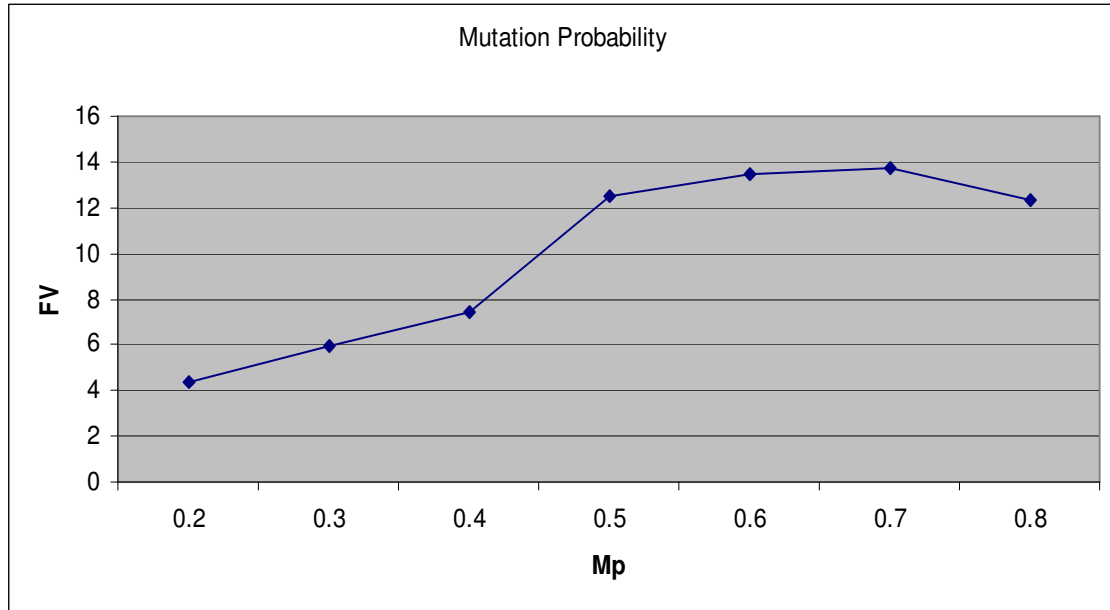


Figure 2.11 C_p -FV Involvement

Table 2.3 Results of C_p -FV Involvement examination

C_p		0.3	0.4	0.5	0.6	0.7	0.8	0.9
FV	If_Better	10.633	13.128	17.794	24.197	33.235	34.081	39.081
	Always	9.624	11.875	15.783	20.232	27.317	32.377	38.801

Figure 2.12 M_p -FV InvolvementTable 2.4 Parameters of training for M_p -FV Involvement examination

S_t	Steady_State	C_p	M_p	P_s	C_t	Replace	T_c	FV
Tournament	False	0.5	0.8	100	two-point	If_better	FV	25

2.8.3 Importance of Elitism

First two individuals which have highest fitness value are added into next generation directly with elitism. The strongest individuals are not lost by this way. These individuals are under protection against every operations like mutation, crossover. To investigate importance of elitism 100 trainings are applied. 50 are with elitism and 50

are without that. To examine that, tournament selection is used as parent selection technique. Effect of group size for tournament selection is investigated, too. 5 different group sizes are taken, these are 10,15,20,25 and 30. For each group size 10 trainings are applied and average of these is taken. Figure 2.13 shows result of investigation as a graph. While observation is done, parameters that are shown in Table 2.4 are used. Graphic that is shown in Figure 2.13 shows effects of G_s and Elitism on GN. FV is used as termination criterion. FV is 25 for that investigation. Blue line at the graphic represents classification with elitism and pink line represents classification without elitism. This graphic proves that importance of elitism. Elitism provides to reach expected FV with smaller GN. G_s effects GN substantially. Too Small G_s s and too big G_s s increases GN as seen in Figure 2.13. 25 is the most suitable G_s for that classification.

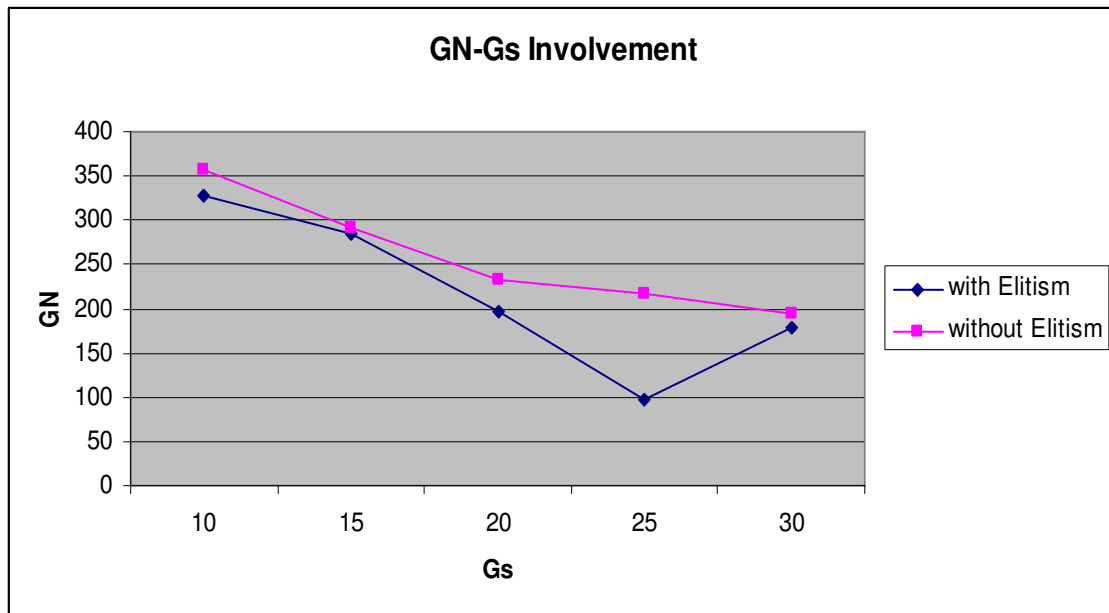


Figure 2.13 GN- G_s Involvement

2.8.4 Population Size

Population size (P_s) is other argument that effects classification with genetic algorithm. To show effect of P_s 50 classification operations are done. 5 different P_s which are between 25 and 250, are taken and FVs of these are observed. 10

classifications are done for each P_s and average of these operations is taken. That investigation is shown in Figure 2.14 and Figure 2.15. The more P_s increases, the more FV increases for fixed GN (Figure 2.14) but also the more training time increases, too (Figure 2.15). Parameters that are fixed during the training are listed in Table 2.5.

Table 2.5 Parameters of training for P_s -FV Involvement examination

S_t	Steady_State	Elitism	C_p	M_p	C_t	Replace	T_c	GN
Best	False	False	0.7	0.6	two-point	If_better	GN	50

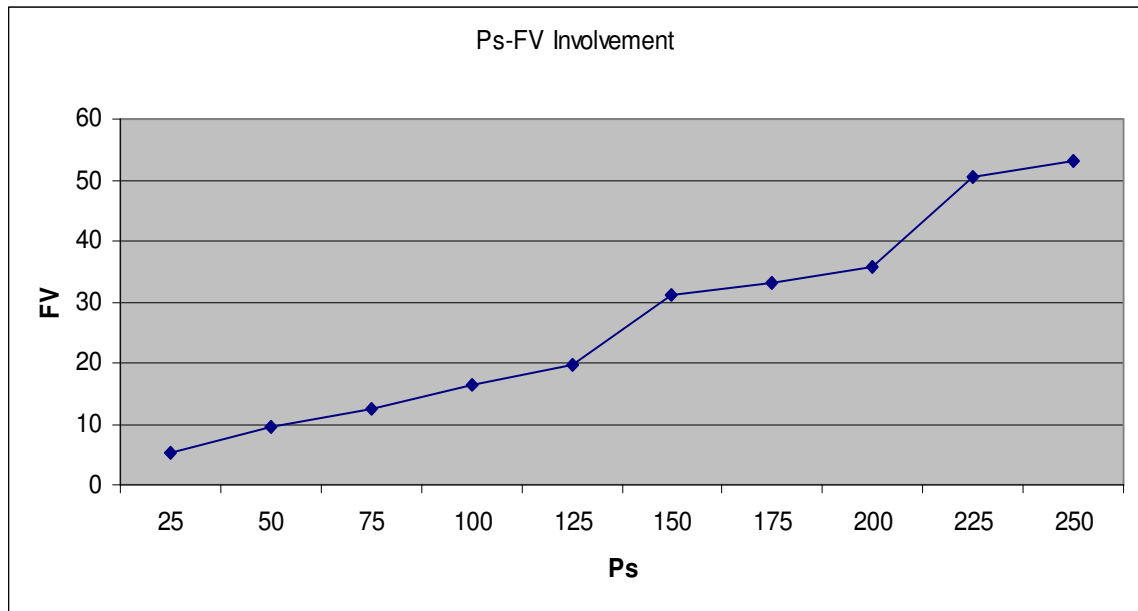


Figure 2.14 P_s -FV Involvement

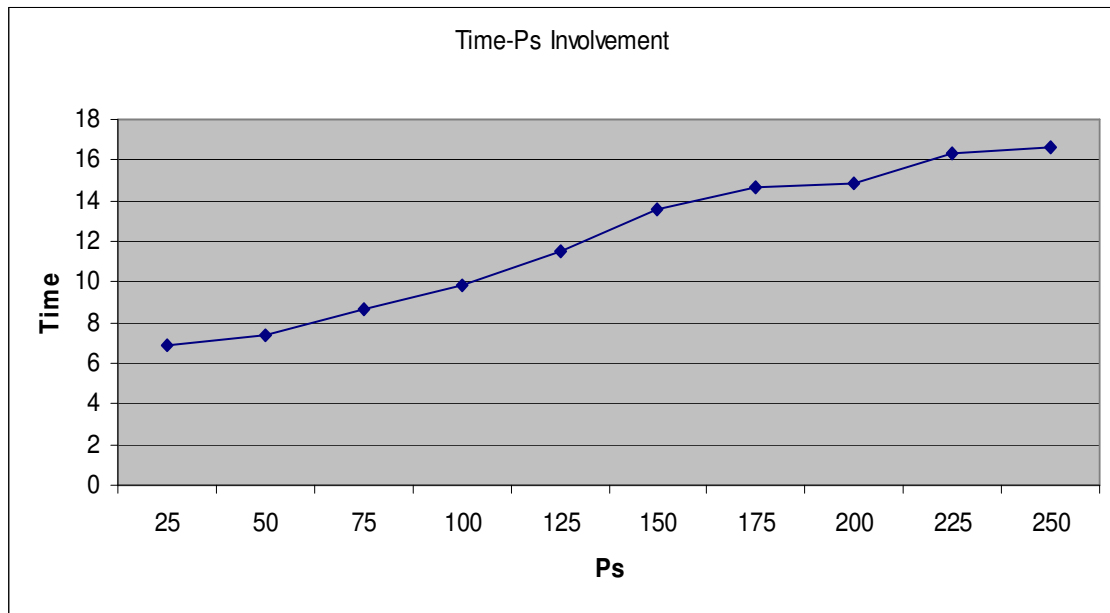


Figure 2.15 Time-Ps Involvement

Table 2.6 Results for P_s – FV Involvement

Ps	FV	Time(sn)
25	5.1	6.93
50	9.56	7.39
75	12.47	8.62
100	16.48	9.85
250	52.99	16.58

2.8.5 Traditional vs. Incremental GA

In this section, the performances of traditional GA and incremental GA are compared through the experiments based on the real-world data set Nursery. The implements of both algorithms were run with various parameter settings and every classification problem has been solved 10 times where average minimum costs were calculated. Consequently, 120 different experiments were handled for this purpose.

In the first case, 150 patterns are removed from the Nursery dataset and training operation was done for that dataset. Fixed parameters that were used for training operation are shown in Table 2.7. After training operation specific model is created. After that operation 150 patterns are added into dataset as new patterns. Model that is created before is used for training operation, and normal training operation is applied without model usage, differences of FVs are shown in Figure 2.16. Each operation is done when GN is 25, 50 and 75, and 6 operations for each. When GN is 25, FV difference is the highest, the more GN increases, the more FV differences decrease.

In the second case, termination criterion was changed from GN to FV. In this case, initial parameters were assigned with the values listed in Table 2.8 and training operations were repeated with different FV values 20, 25 and 30. According to the results shown in Figure 2.17, incremental GA reached to the expected FV at least 3 times faster than traditional GA. So, incremental GA provides to access expected fitness value with lower generation number.

Table 2.7 Parameters of training for incremental GA when termination criterion is GN.

S_t	Steady_State	Elitism	C_p	M_p	C_t	Replace	T_c	P_s
Best	False	False	0.7	0.6	two-point	If_better	GN	226

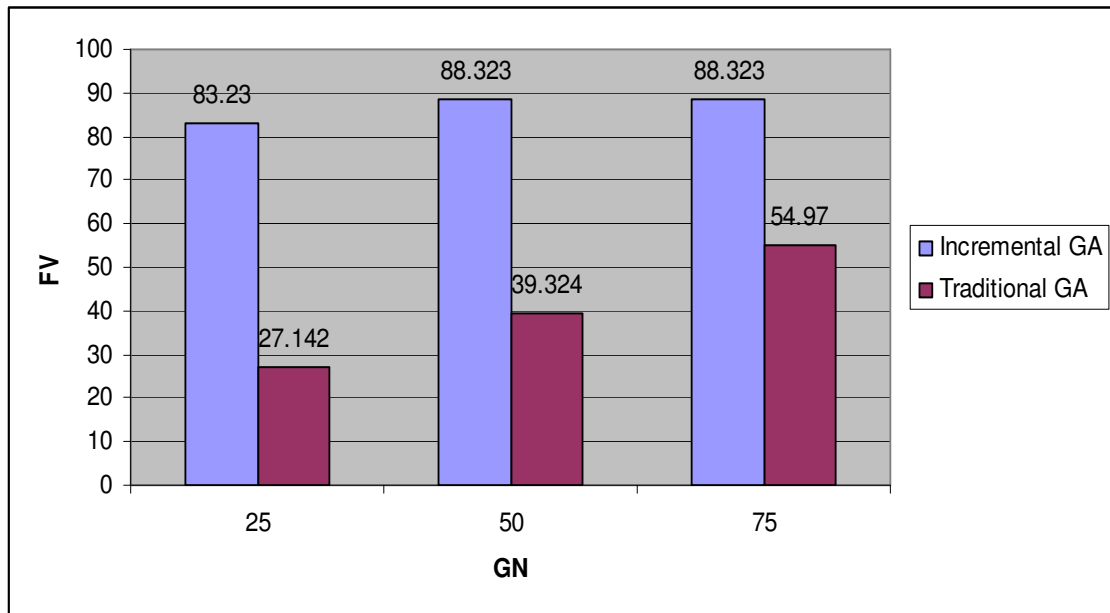


Figure 2.16 Comparison of Traditional GA and Incremental GA according to the average FVs with various GN parameter settings

Table 2.8 Parameters of training for incremental GA when termination criterion is FV.

S_t	Steady_State	Elitism	C_p	M_p	C_t	Replace	T_c	P_s
Best	False	False	0.7	0.6	two-point	If_better	FV	226

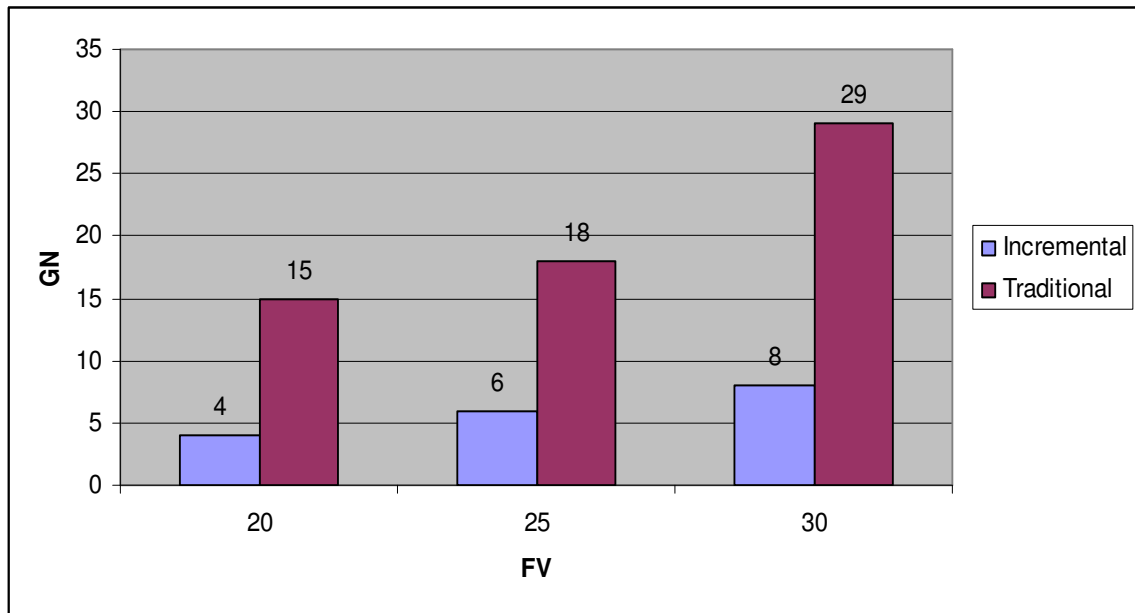


Figure 2.17 Comparison of Traditional GA and Incremental GA according to GN values with various FV parameter settings

2.8.6 Classification Accuracy

The classification accuracy is evaluated by the error rate that is the ratio of the total number of correctly classified samples by the trained models in all generated test. For example, the dataset could be randomly divided into two portions, with 70 percent of data in the training set and 30 percent in the validation (test) set. After training operation is performed on the training set, classification accuracy rate is computed on the test set.

Commonly used validation techniques for classification are simple validation, cross validation, n -fold cross validation, and bootstrap method. (Kim, 2009) In my experiments, classification accuracy is estimated by using 5-fold cross validation technique. In n -fold cross validation technique, the data set is divided into n subsets, and the method is repeated n times. Each time, one of the n subsets is used as the test set and the other $n-1$ subsets are put together to form a training set. Then the average error

across all n trials is computed. I used this technique because it matters less how the data gets divided.

In my experiments, the highest classification accuracy (89%) is obtained when the input parameters are assigned with the values listed in Table 2.9.

Table 2.9 Specific model that is used for classification

S_t	Steady_State	Elitism	C_p	M_p	C_t	Replace	T_c	P_s	GN	FV
Best	False	False	0.7	0.6	two-point	If_better	GN	226	50	101,65

Table 2.10 Rules that are used for classification

Rules	Class
If (parents=usual) and (has_nurs=proper) and (form=complete) and (children=1) and (housing=convenient) and (finance=convenient) and (social=nonprob or social=slightly_prob) and (health=recommended)	recommend
If (parents=usual or parents=pretentious) and (has_nurs=proper or has_nurs=less_proper or has_nurs=improper) and (health=recommended or health=priority)	priority
If (health=not_recom)	not_recom
If (parents=usual or parents=pretentious) and (has_nurs=proper or has_nurs=less_proper) and (form=complete or form=completed or form=incomplete) and (children=1 or children=2) and (housing=convenient or housing=less_conv) and (social=nonprob or social=slightly_prob) and (health=recommended)	very_recom
If (parents=pretentious or parents=great_pret) and (has_nurs=improper or has_nurs=critical or has_nurs=very_crit) and (health=recommended or health=priority)	spec_prior

2.9 Interface

All operations are applied by using a tool that is developed for classification by using genetic algorithm. This tool is developed in Visual Studio .Net 2008, using C# as programming language.

All parameters for classification are entered as inputs. This provides user control and models that are created with different GA parameters, comparisons.

This tool provides model training, comparison, testing, correctness test and incremental GA. Figure 2.18 shows interface.

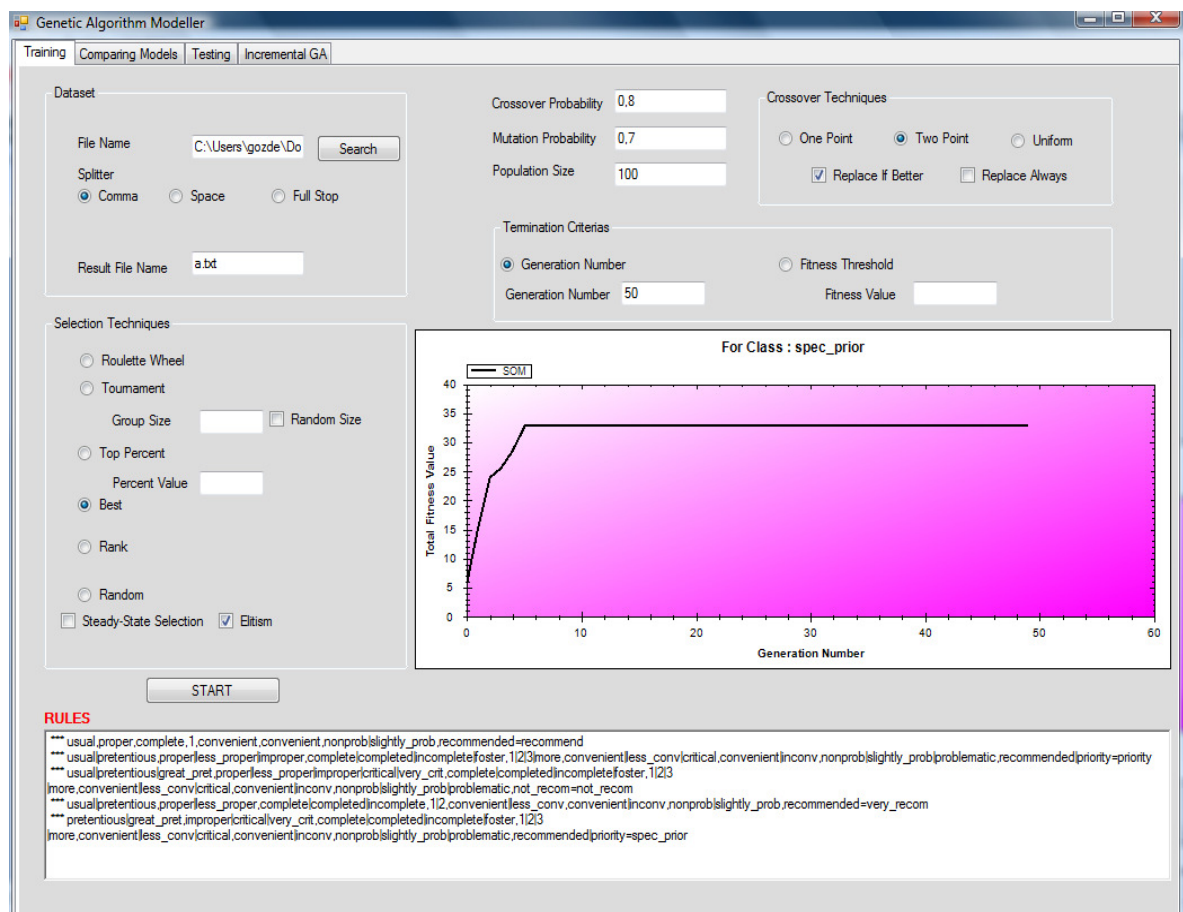


Figure 2.18 Genetic Algorithm Modeller Interface

2.9.1 Training

This function of the tool provides classification. All parameters for the genetic algorithm are entered manually to give all controls to the users. There are 7 main parts of the page. The first part is for dataset. Figure 2.19 shows that part.

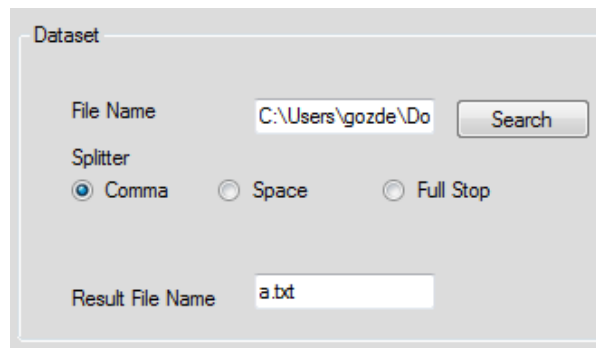


Figure 2.19 Entering dataset

File name of the dataset is entered manually or by using search button. After that, splitter between attributes in the dataset is selected. There are three choices; comma, space and full stop.

After these operations result file name is entered. All information and results of that training operation are saved in that file. This is called 'model' of the classification. This model is used later for many operations.

There are 6 parent selection techniques in the interface as it is shown in Figure 2.20.

Selection Techniques

☐ Roulette Wheel

☐ Tournament

Group Size ☐ Random Size

☐ Top Percent

Percent Value

☒ Best

☐ Rank

☐ Random

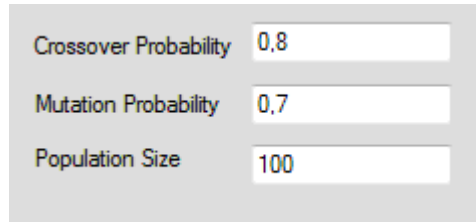
☐ Steady-State Selection ☒ Elitism

Figure 2.20 Selection techniques

Alternatives for selection techniques are Roulette Wheel, Tournament, Top Percent, Best, Rank and Random. If the user select tournament, then she/he should determine group size for tournament. There are two alternatives for group size. User can enter group size manually, or she/he can select random size. If random size is selected, group size will be determined randomly. If top percent is selected as a parent selection technique, then percent value should be entered manually. For example there are 1000 individuals in the population. User enters 30 for the percent value. In that situation parents are selected between first 300 individuals in the population.

After selection techniques are determined, there two alternatives which can improve effectivity of the training operation. These are *Steady-State Selection* and *Elitism*. User can select one of them or not, it is optional. If Steady-State Selection is selected, then first %30 of population which have high fitness value, are transferred into next generation without any operation that can change them. If Elitism is selected, then first individual which have highest fitness value, is transferred into next generation directly. Steady-State Selection and Elitism can not be selected together.

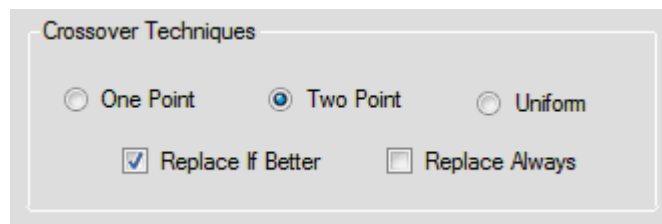
GA parameters such as *Crossover Probability*, *Mutation Probability* and *Population Size* are determined by user shown in Figure 2.21.



Crossover Probability	0.8
Mutation Probability	0.7
Population Size	100

Figure 2.21 GA parameters

Crossover probability and mutation probability should be between 0 and 1.



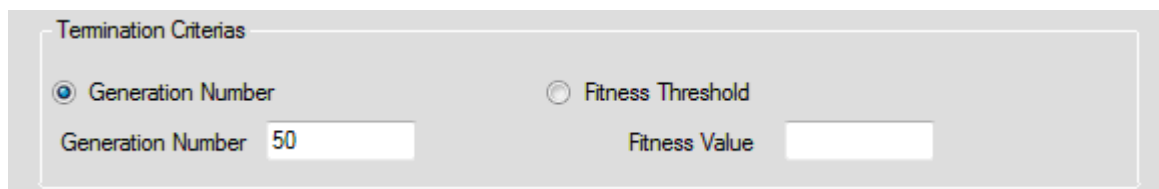
Crossover Techniques

☐ One Point
 ☒ Two Point
 ☐ Uniform

☒ Replace If Better
 ☐ Replace Always

Figure 2.22 Crossover techniques

There are three crossover techniques alternatives for classification with GA. These are *One Point*, *Two Point* and *Uniform*. These are illustrated in Figure 2.22. After crossover techniques, there are two choices that can improve performance of training. These are *Replace If Better* and *Replace Always*. Selection these, is optional like Elitism and *Steady_State_Selection* in Figure 2.20.



Termination Criterias

☒ Generation Number
 ☐ Fitness Threshold

Generation Number 50
 Fitness Value

Figure 2.23 Termination criteria

Termination Criteria are in user control like other GA parameters. There are two termination criteria, *Generation Number* and *Fitness Value*, see Figure 2.23. If generation number is selected as termination criterion, then generation number should be determined. For example for the training that is shown in Figure 2.23, training will stop after 50 new generations. If the termination criterion is Fitness Threshold, then user should determine Fitness Value. Training terminates when fitness value of population reaches that user defined value.

During training operation, information of generation number and fitness value of operation is shown by using graphic that is illustrated in Figure 2.24.

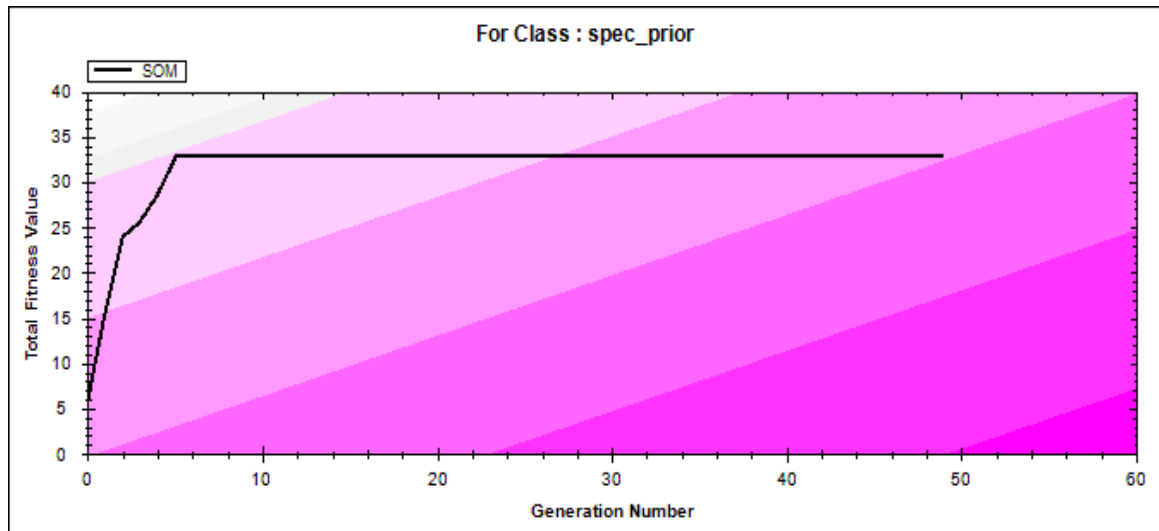


Figure 2.24 Graphic

This graphic is drawn by using Zed Graph. For each class this graphic is drawn. For example for nursery dataset, this graphic is shown to the users for 5 times, because there are five distinct classes in the dataset. This graphic provides to show when model reaches at maximum FV. For example for the training that is shown in Figure 2.24, maximum FV is reached nearly after 5 generation numbers, so that provides to give information to the user that 50 generation number is too big for that training.

RULES

```

***usual.proper.complete,1.convenient.convenient.nonprobslightly_prob.recommended=recommend
***usual.pretentious.properless_properimproper.complete|completed|incomplete|foster,1|2|3|more.convenient|less_conv|critical.convenient|inconv.nonprobslightly_prob.problematic.recommended|priority=priority
***usual.pretentious|great_pret.properless_properimproper|critical|very_crit.complete|completed|incomplete|foster,1|2|3
more.convenient|less_conv|critical.convenient|inconv.nonprobslightly_prob.problematic.not_recom=not_recom
***usual.pretentious.properless_proper.complete|completed|incomplete,1|2.convenient|less_conv.convenient|inconv.nonprobslightly_prob.recommended=very_recom
***pretentious|great_pret.improper|critical|very_crit.complete|completed|incomplete|foster,1|2|3
more.convenient|less_conv|critical.convenient|inconv.nonprobslightly_prob.problematic.recommended|priority=spec_prior

```

Figure 2.25 Rules

For each class, rules are listed as output like in Figure 2.25 ‘,’ means *and*, ‘|’ means *or*, and ‘=’ stands for *then*. All of these rules are saved in model file to use later, too.

2.9.2 Comparing Models

This function of the tool provides to compare models according to their performances, and results are shown by using bar chart. Figure 2.26 shows interface of ‘Comparing Models’.

There is a datagridview that illustrates all information of the models which are compared. Infinite number of models can be compared. In datagridview; model name, selection type, information about steady_stead selection and elitism, crossover and mutation probabilities, population size, crossover type, replace_if_better and replace always information, termination criteria, generation number and average fitness value are listed.

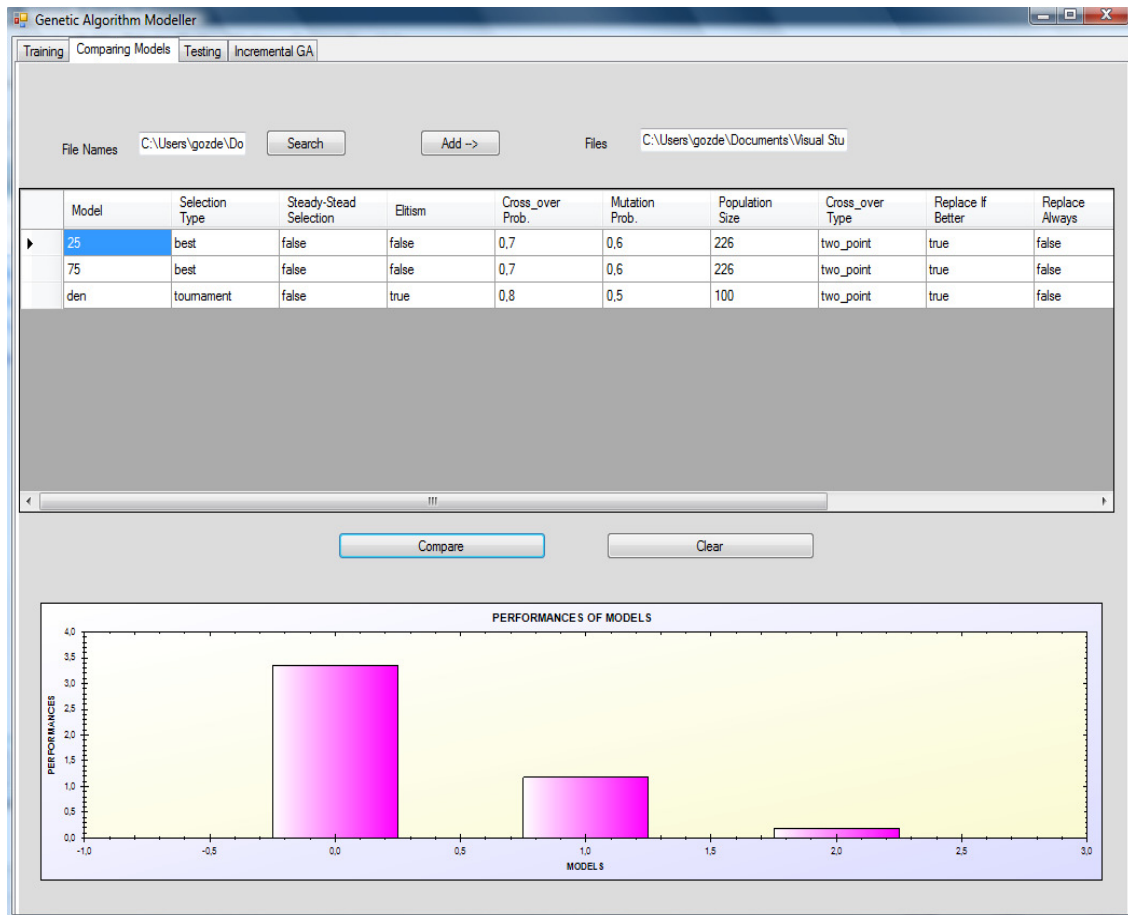


Figure 2.26 Comparing models

First of all; models are selected by 'Search' button, and then added into list in the datagridview by using 'Add->' button. After these operations models are compared with 'Compare' button. As it can be seen from the Figure 2.26, performances of all models are illustrated by using Bar Chart that is drawn with Zed Graph. For example for that comparison that is shown in Figure 2.26, first model which is called '25' has the highest performance and model '75' has greater performance than model 'den'. Reason of that result is shown in the datagridview. Look at Figure 2.27, that shows other part of the datagridview which is not seen in the Figure 2.26.

Model	Cross_over Prob.	Mutation Prob.	Population Size	Cross_over Type	Replace If Better	Replace Always	Termination Criteria	Generation Number	Average Fitness
25	0,7	0,6	226	two_point	true	false	Generation Number	25	83,53166643518...
75	0,7	0,6	226	two_point	true	false	Generation Number	75	88,32370347222...
den	0,8	0,5	100	two_point	true	false	Fitness Threshold	130	25

Figure 2.27 Information of models

Performance is directly proportional to the *fitness* and inversely proportional to the *generation number*. Performance is calculated as it is declared in Definition 9. Fitness value of the first model is '83.53' and fitness value of the second model is '88.32'. If just fitness values were looked for the performance calculation, then second '75' model should have higher performance than '25' model. But when generation numbers of models are looked, then it can be seen that '75' model has three times of generation number of '25' model. So '25' model has the highest performance. 'den' model has the smallest fitness value and also highest generation number. So that model has the smallest performance between these models. User can see the reason of that result by looking information of the models. For example population sizes of first two models are '226', and 'den' model has '100' population size. So it can be said that the more population size increases, the more performance increases. The other difference between first two models and the third model is crossover and mutation probabilities. So all of reasons that effect performance, can be seen with comparing function of the tool.

If any other models are wanted to compare then 'Clear' button is used to clear datagridview and graph.

2.9.3 Testing

This part of tool provides to classify patterns which don't belong any classes. Testing page includes two parts. First part is for classifying, and the second part is to test how model is correct. Figure 2.28 shows testing page.

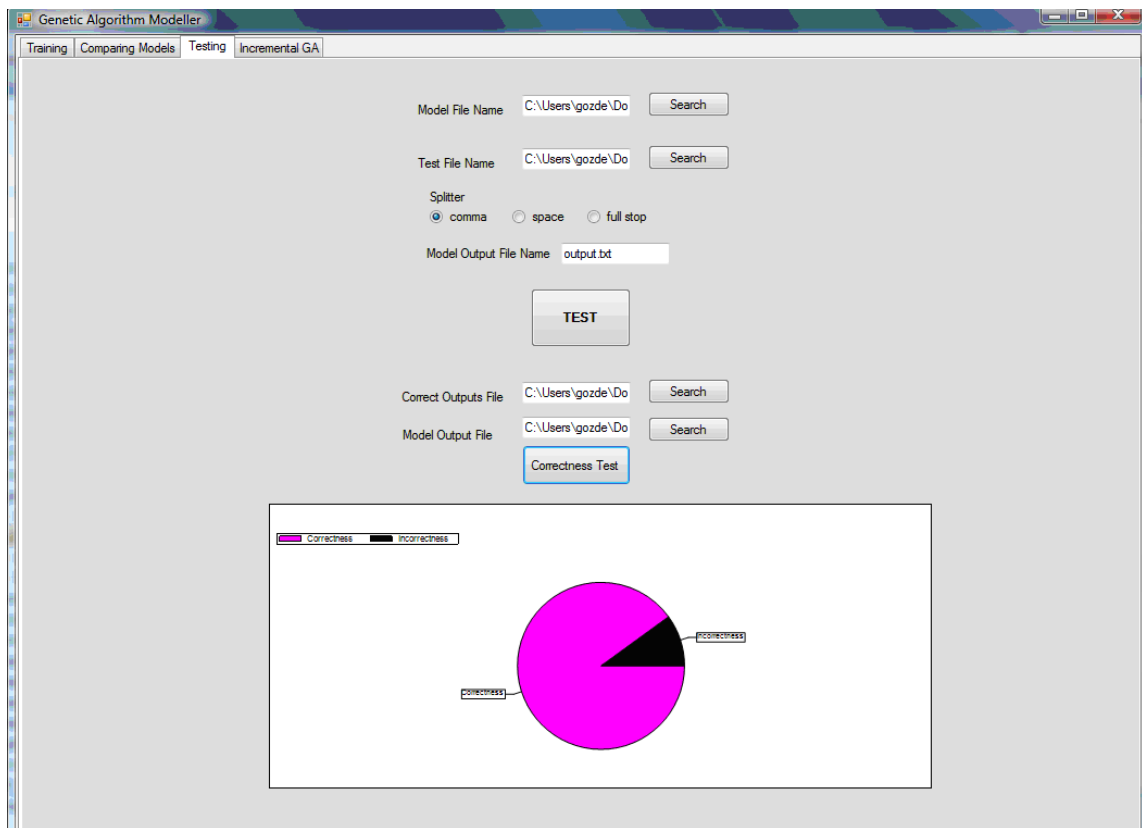


Figure 2.28 Testing

Model file name which includes classification rules is entered in the first textbox. Test file name that includes patterns that are wanted to classify is written in the second text box. Splitter that splits attributes in the test file is selected, 'comma', 'space' or 'full stop'. Model output file name should be written in the third text box. Class names of all patterns are written in that model output file. After pressing 'Test' button, class names that are found for patterns are written in the 'output.txt' file for that example that is shown in Figure 2.29.

Model File Name C:\Users\gozde\Do Search

Test File Name C:\Users\gozde\Do Search

Splitter
☒ comma ☐ space ☐ full stop

Model Output File Name output.txt

TEST

Figure 2.29 Classification

To test correctness of model, second part of the page is used. To use that function of the tool, user should have file that includes correct class name of the patterns. Correct outputs filename that includes correct class name of patterns is entered in the 'Correct Outputs File' text box, and model output file name is entered in the 'Model Output File' text box. After pressing 'Correctness Test' button, correctness ratio of the model is shown by using pie chart that is drawn with Zed Graph as it is shown in Figure 2.30.

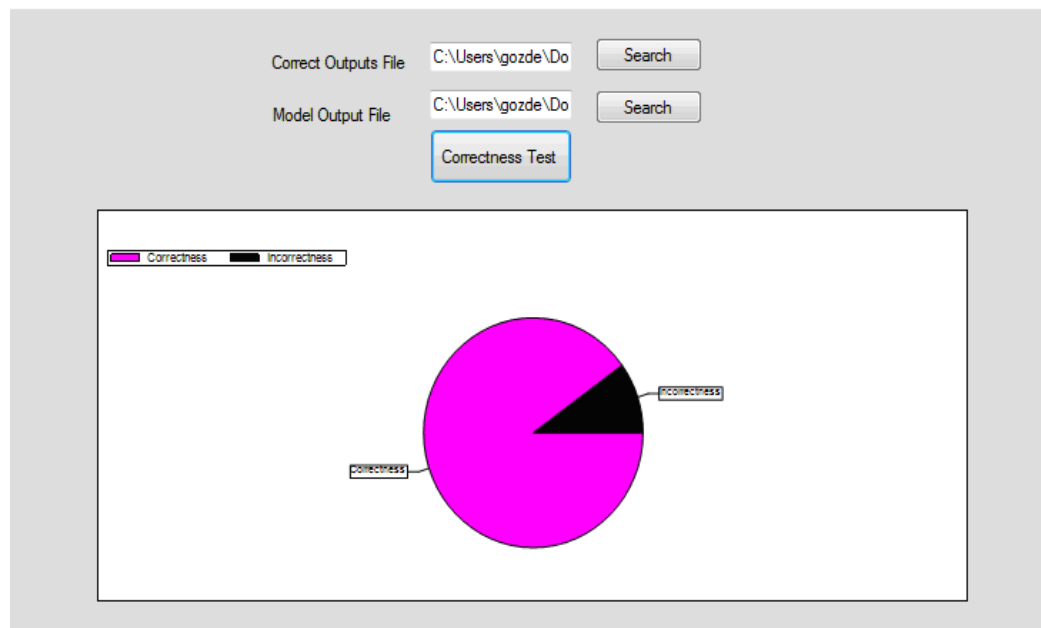


Figure 2.30 Correctness test

2.9.4 Incremental GA

This function of the tool provides training that needs less classification time than traditional training. This function is used for datasets that are updated regularly.

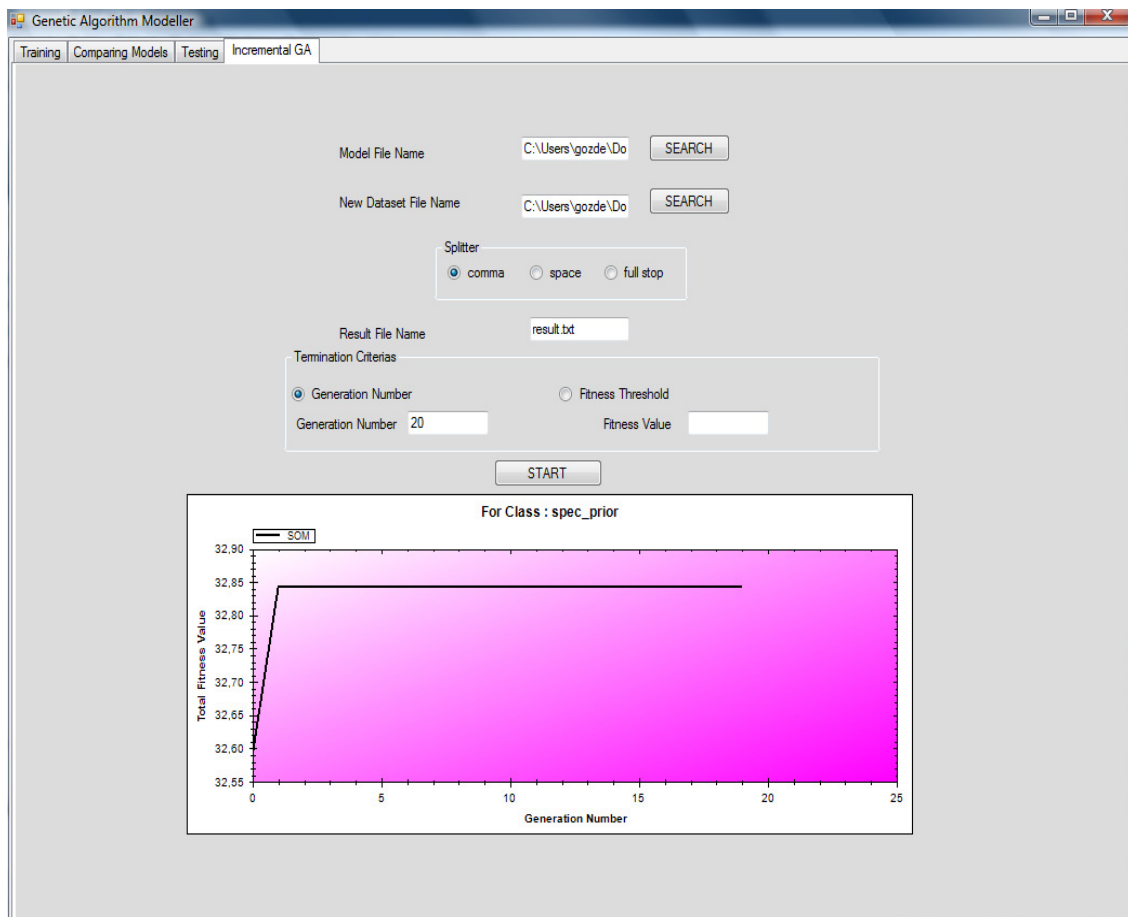


Figure 2.31 Incremental GA

To achieve Incremental GA, name of model which is created for previous dataset, is entered in the 'Model File Name' text box. This model includes classification rules that are found with traditional GA classification for previous dataset. Dataset file name that includes new added patterns is entered 'New Dataset File Name' text box that is shown in Figure 2.32.

Figure 2.32 Incremental GA I

Symbol that splits attributes in the dataset is selected between there choices; ‘comma’, ‘space’ and ‘full stop’. And then result file name which all information about training will be written in, is entered in the ‘Result File Name’ text box.

Another advantage of that operation is that there is no need to write GA parameters for training. Because all needed GA parameters are in the *model file*.

Termination Criteria should be chosen to terminate training operation. There are two choices as in traditional training; *Generation Number* and *Fitness Value* as in Figure 2.33. If *Generation Number* is selected as termination criterion, then generation number should be written in ‘Generation Number’ text box. If *Fitness Value* is selected, then fitness value should be entered in ‘Fitness Value’ text box.

Figure 2.33 Incremental GA II

After pressing ‘START’ button training operation starts. Graph that is shown in Figure 2.34 is drawn during training. This graph shows situation of operation, and gives information to the user about FV and GN for each class as in traditional training operation.

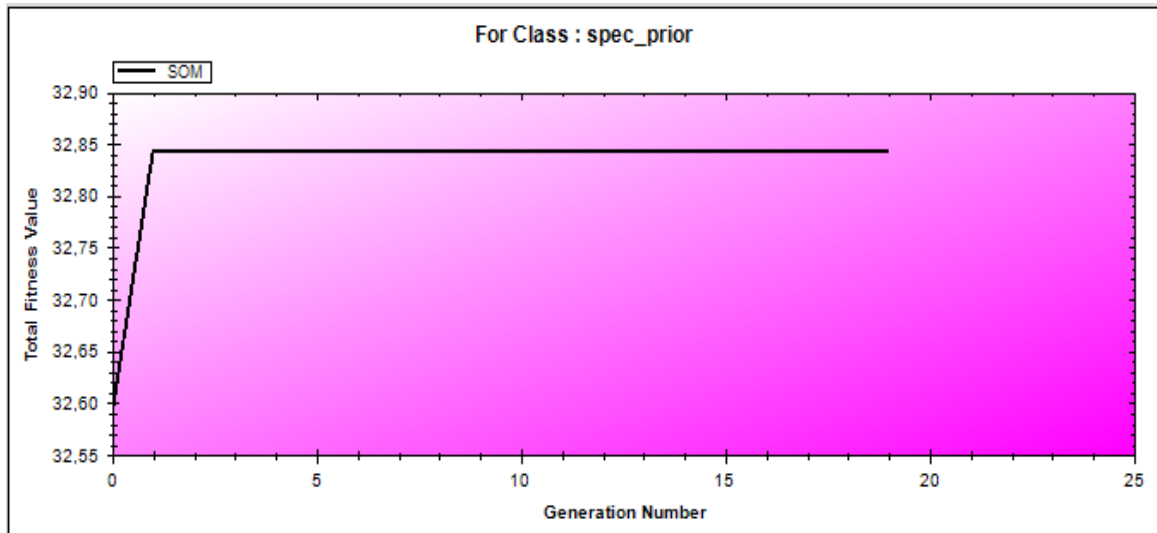


Figure 2.34 Graph for Incremental GA

In that example that is illustrated in figures above, firstly nursery dataset that includes 12810 patterns is trained and a model is created for that operation. After that, 150 new patterns are added in dataset. If there is not that Incremental GA function of the tool, then traditional training operation would be applied, and this would cause waste of time. If Figure 2.34 is looked, it can be seen that FV for spec_prior class starts from 32.60, not 0. Because initial population is not created fully randomly, classification for that exists in model file, is added in initial population.

CHAPTER THREE

NEURAL NETWORK

3.1 Related Works

In recent years, many studies which NNs have been applied for classification problem have been done. Mazurowski , Habas, Zurada, Lo, Baker and Tourassi (2007), investigate the effect of class imbalance in training data when developing neural network classifiers for computer-aided medical diagnosis. Molnár, Keserű, Papp, Lőrincz , Ambrus and Ferenc Darvas (2006) developed a NN based classification approach using cytotoxicity data measured for 30,000 compounds to predict cytotoxicity. Yu and Zhu (2009) combined neural networks and semantic feature space for email classification. Manevitz and Yousef (2006) developed one-class document classification by using Neural Networks. Banerjee, Kiran, Murty and Venkateswarlu (2008) presented an ANN for classification and identification of *Anopheles* mosquito species based on the internal transcribed spacer2 (ITS2) data of ribosomal DNA string. Übeyli (2008) used combined NN model to guide model selection for classification of electroencephalogram (EEG) signals.

3.2 Neural Network

NN is a system that includes units which have small amount of local memory. These units connect each other with more than one communication channel. These channels carry out numerical data. Each unit process its local data, and units run asynchronously.

3.2.1 Simple Single Unit Network

Simple single unit network includes input, weights, nucleus, activation and output as it is shown in Figure 3.1.

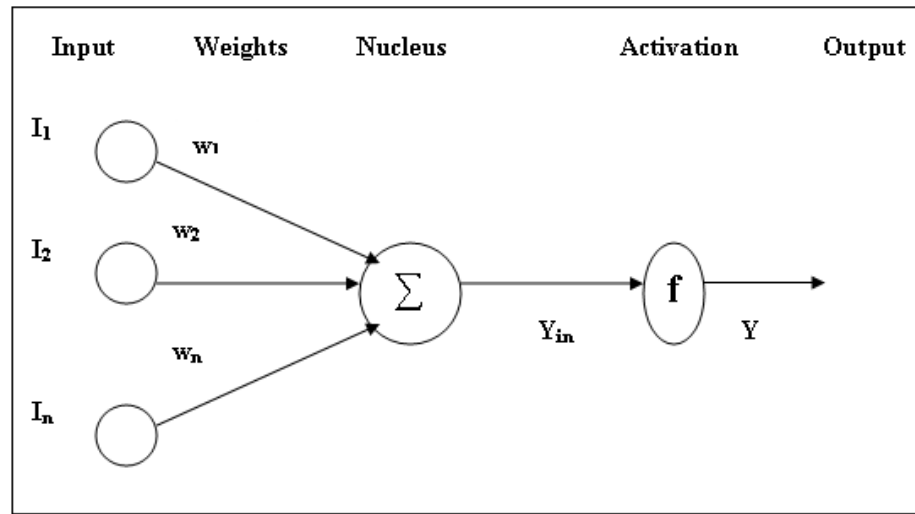


Figure 3.1 Simple single unit network

I_n is input of network, w_n is weight of I_n . Nucleus includes a summation function that calculate weighted sum of inputs. Most widely used summation function is illustrated below.

Definition 10 Weighted sum of inputs calculated as;

$$Y_{in} = \sum_{i=1}^n I_n w_n$$

where I_n is input value, and w_n is the weight.

Y_{in} is the input of activation function, f . Activation function provides to process input that is calculated with summation function, and to calculate output of the network. There are many activation functions. In that study Sigmoid, Gaussian, Identity, Unit Step, Piecewise Linear and Hyperbolic Tangent functions are used as activation functions.

3.2.2 Activation Functions

3.2.2.1 Sigmoid Function

This is the most widely used activation function in NN. Sigmoid function gives continuous results to the inputs. Results are not discrete. This function is suitable for the problems which sensitive evaluation should be applied. Result of the sigmoid function is between 0 and 1.

Definition 11 Sigmoid function is;

$$f(x) = \frac{1}{1 + e^{-\beta(x+b)}}$$

where β is gradient, x is input and b is the bias.

3.2.2.2 Gaussian Function

Gaussian function provides easier to prediction of the behaviour of the net when the input patterns differ strongly from all teaching patterns.

Definition 12 Gaussian function as activation function is

$$f(x) = \frac{1}{\sqrt{2\pi\beta}} e^{\frac{-(x-\mu)^2}{2\beta^2}}$$

where β is gradient, x is input and μ is the learning rate

3.2.2.3 Identity Function

This function is noneffective function.

Definition 13 Identity function is;

$$f(x) = x$$

3.2.2.4 Unit Step Function

If input is greater than 0, output is 1, otherwise output is 0. This function can be used for simple problems. This function is not useful for complex problems.

Definition 14 Unit step function is;

$$f(x) = 0 \text{ if } x < 0, f(x) = 1 \text{ if } x \geq 0$$

3.2.2.5 Piecewise Linear Function

Piecewise linear function is combination of sigmoid and unit step functions. This function returns both continues and discrete values. This function returns continues values between 0 and 1, and calculates discrete values 0 or 1.

Definition 15 Piecewise Linear function is;

$$f(x) = \begin{cases} 0 & \text{if } x \leq x_{\min} \\ mx+b & \text{if } x_{\min} < x < x_{\max} \\ 1 & \text{if } x \geq x_{\max} \end{cases}$$

3.2.2.6 Hyperbolic Tangent

Difference of that function from others, that function returns results between -1 and 1.

Definition 16 Hyperbolic Tangent function is;

$$f(x) = (1 - e^{-2x}) / (1 + e^{-2x})$$

3.2.3 Termination Criteria

Training operation in neural network attend until termination criteria are met. There are two termination criteria that are most widely used. These are *minimum error* and *iteration number*. In that study these are used as termination criteria.

3.2.3.1 Minimum Error

As its names, there is a given value that represents minimum error which network should reach that value. Training operation continues until error value of the network reaches given minimum error value. Disadvantage of that method is wrong minimum error value which system can never reach, can be given. In that situation, loop can not terminate.

3.2.3.2 Iteration Number

There is a given number which shows how many iterations will be done. Training attends until iteration number reaches given iteration number. When it reaches limit value, training terminates. Disadvantage of that method, small iteration number can be given as termination criterion. In that situation, training terminates before reaching optimum result. When bigger than network needs to reach optimum solution,

termination number is given, and then loop continues unnecessary after finding optimum solution.

3.2.4 *Neural Network Types*

Neural networks are categorized according to learning strategies and architectures. There are three learning strategies in neural networks. These are *supervised*, *unsupervised* and *reinforcement* learning. In *supervised learning* as its names, prediction is done by using given input/output pairs. Output of the network for current input is compared with the given output, and then error value is calculated, and weights are updated. In *unsupervised learning* there is not any intervention to effect neural network from outside. Correct output values are not given to the network and network classify patterns with comparing to patterns. *Reinforcement learning* is not supervised or unsupervised learning. In that learning strategy correct output is not given, but output of the network is evaluated as true or false.

According to architectures, there are two neural network types. These are *feedforward* and *feedback* neural networks. In *feedforward neural network*, data stream is one way. There is no way for feedbacking. A cell accepts input from only previous cells. *Feedback* neural network provides feed backing, cells accept input from all layers in network.

In that study four neural network types are used for classification operation. These are; *Backpropagation*, *Single Layer Perceptron*, *Multilayer Perceptron* and *Self Organizing Map* Neural Networks.

3.2.4.1 Single Layer Perceptron

This model underlies of the neural network of today. Activation function is used in that model for the first time. There are only input and output layers in that neural network model, there is not any hidden layer. This is a feedforward model. That model doesn't support feedback in the network. Figure 3.2 illustrates architecture of the single layer perceptron.

Single layer perceptron used supervised learning.

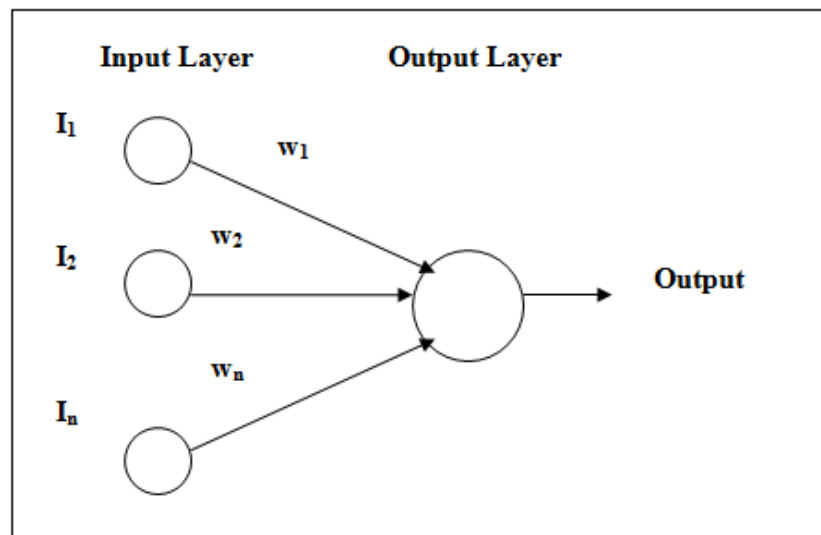


Figure 3.2 Single Layer Perceptron

This model is suitable for simple problems, not for complex problems. Algorithm of single layer network classification is illustrated in Figure 3.3.

1. Initialize weights randomly
2. For each pattern repeat 2-8
3. Take input pattern to the input layer
4. Calculate weighted sum of inputs, \sum
5. Give \sum to the output neuron as input
6. Calculate output of the network using activation function in output neuron
7. Calculate error of the network
8. Update weights
9. Repeat these steps until termination criterion is met

Figure 3.3 Single layer perceptron algorithm

3.2.4.2 Multilayer Perceptron (MLP)

As its names, difference of MLP from single layer perceptron is hidden layer. MLP runs like single layer perceptron. MLP is better than single layer perceptron because of hidden layers, worse than backpropagation because of omission of feedback. MLP uses supervised learning.

Architecture of MLP is shown in Figure 3.4. There are three basic layers in the MLP architecture; input layer, hidden layer and output layer. Hidden layer can be more than one.

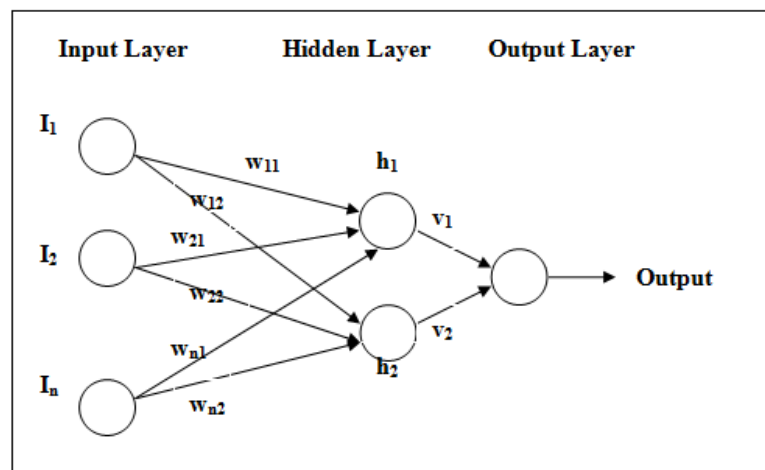


Figure 3.4 Multilayer perceptron architecture

3.2.4.3 Backpropagation Neural Network

Backpropagation is a kind of neural network, that can be applied all kinds of problems easily. Backpropagation is most widely used neural network type. Difference from other NN types that are declared before, is using feedback. There are input, hidden and output layers in Backpropagation NN. There can be more than one hidden layers. Standard three layer Backpropagation architecture is illustrated in Figure 3.5.

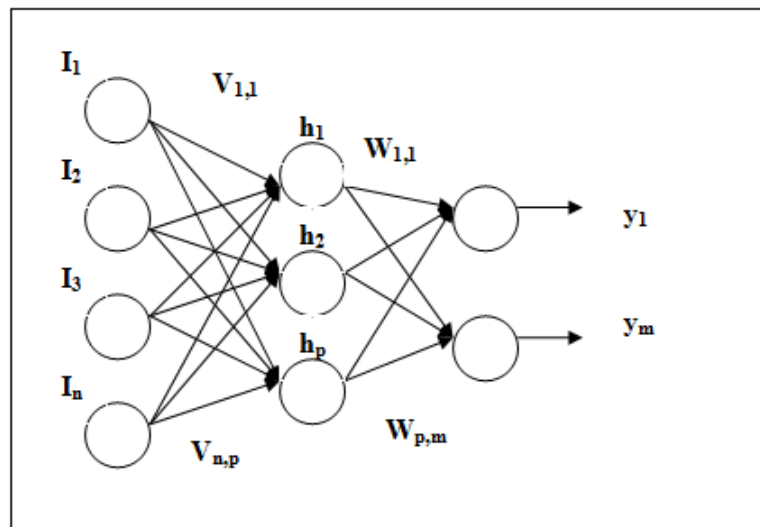


Figure 3.5 Backpropagation architecture

Backpropagation algorithm is shown in Figure 3.6.

```

Initialize weights
 $0 < V_{ij} < 1, 0 < W_{jk} < 1$ 
Repeat
  For each patterns
    Take inputs and transmit next layer,  $I_{\text{output}} = I_m$ 
    For each hidden layer
      For each neuron in hidden layer
        Calculate summation for neuron
 $h_{\text{input}_j} = \sum I_{\text{output}} * V_{ij}$ 
        Calculate output of current neuron
 $h_j = f(h_{\text{input}_j})$ 
      End for
    End for
    For each neuron in output layer
      Calculate summation for neuron
 $y_{\text{input}_k} = \sum h_j * W_{jk}$ 
      Calculate output of neuron
 $y_k = f(y_{\text{input}_k})$ 
    End for
    Calculate error of output layer
 $\hat{o}_k = (t_k - y_k) * f'(y_{\text{in}_k})$ 
    Find exchange value for weights between output-hidden layers
 $\Delta w_{jk} = \mu * \hat{o}_k * z_j$ 
    For each hidden layer
      Calculate error
 $\hat{o}_{\text{in}_j} = \sum \hat{o}_k * w_{jk}$ 
    End for
    Find error value that effects input layer
 $\hat{o}_j = \hat{o}_{\text{in}_j} * f'(z_{\text{in}_j})$ 
    Find exchange value for weights between hidden-output layers
 $\Delta v_{ij} = \mu * \hat{o}_j * x_i$ 
    Update all weights
 $w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}$ 
 $v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}$ 
  End for
Until termination criteria are met

```

Figure 3.6 Backpropagation algorithm

3.2.4.4 Kohonen's Self Organizing Map (SOM)

Difference of that model from others that are explained before, is unsupervised. Kohonen uses associative memory in that model. That means, intense between two objects and neighbourhood relationship are taken notice to classify.

There are group midpoints that are called clusters in SOM. If suitable substitutes can be chosen, this population can be presented by those substitutes. In other words, a population that includes 'm' components can be divided 'n' classes, and those n classes can be presented by origin of those groups. To achieve that, first 'n' patterns are chosen as representative of n groups and all components are learned according to these points. Input pattern is compared with all clusters and suitable cluster is chosen for that input.

SOM algorithm is illustrated as pseudo code in Figure 3.7.

```

Initialize weights
   $0 < w_i < 1$ 
Initialize learning rate
   $0 < \mu < 1$ 
Repeat
  For each patterns
    For each cluster (j)
      Calculate distance
         $distance(j) = \sum (w_{ij} - x_i)^2$ 
      End for
      Find cluster that has minimum distance
         $index = \min \{j, uzaklik(j)\}$ 
      Update weights for that cluster and neighbours
         $w_{i,index}(new) = w_{i,index}(old) + \mu * (x_{i,index} - w_{i,index}(old))$ 
    End for
    Sensitize learning rate
       $\mu = \mu * 0.6$ 
    Reduce neighbourhood rate
  Until error  $\leq$  minimum error

```

Figure 3.7 SOM algorithm

3.3 Approach

3.3.1 Incremental Approach

This study provides incremental neural network classification. Some datasets are updated regularly. In that situation classification operation should be applied to all dataset for each update operation. But it is expensive and this causes waste of time. To prevent this, an incremental NN method is developed. With that method instead of training whole dataset, only part of dataset that includes new patterns is trained. Training time decreases with that method and so performance of the model increases. To achieve incremental NN, model that is illustrated in Figure 3.10 is used. These specific models include weights that are obtained previous training operation. At the beginning of the training operation weights are not initialised randomly, initialised with values that exist in the model file. Figure 3.8 shows steps of Incremental NN operation.

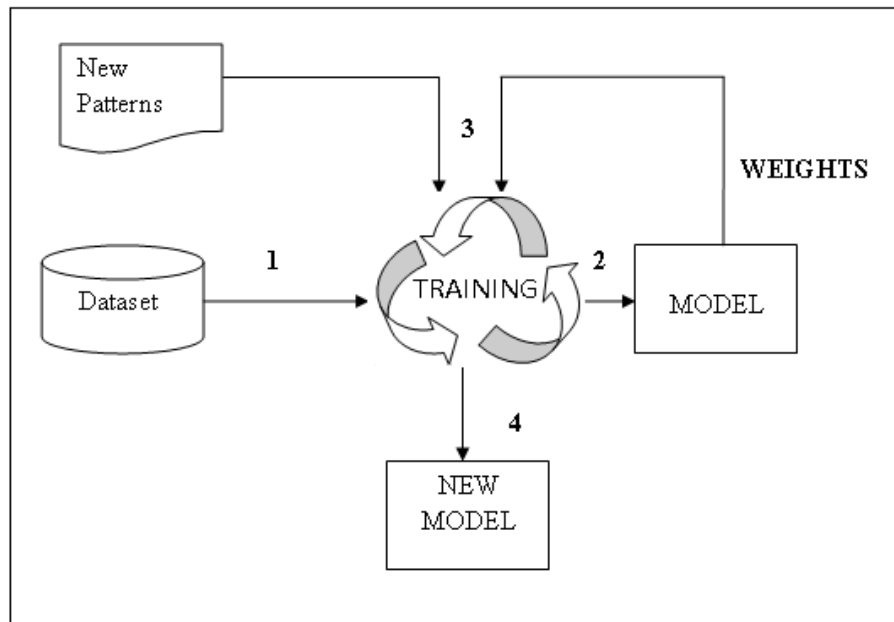


Figure 3.8 Incremental approach for handling new patterns

At the beginning, initial dataset is trained by using classifier tool. After that operation a specific model that includes all inputs and outputs of training, is created. When new patterns add in dataset, just these patterns are trained by using weights that exist in the model. So we can train new dataset just by training new patterns. Backpropagation algorithm for incremental approach is shown in Figure 3.9. For each NN types, algorithm changes like backpropagation algorithm for incremental.

```

Read classifier model previously constructed
Initial weights ( $V_{ij}, W_{jk}$ ) with weight values in the classifier model sequentially
Repeat
  For each new patterns
    Take inputs and transmit next layer,  $I_{\text{output}} = I_m$ 
    For each hidden layer
      For each neuron in hidden layer
        Calculate summation for neuron
         $h_{\text{input}_j} = \sum I_{\text{output}} * V_{ij}$ 
        Calculate output of current neuron
         $h_j = f(h_{\text{input}_j})$ 
      End for
    End for
    For each neuron in output layer
      Calculate summation for neuron
       $y_{\text{input}_k} = \sum h_j * W_{jk}$ 
      Calculate output of neuron
       $y_k = f(y_{\text{input}_k})$ 
    End for
    Calculate error of output layer
     $\partial_k = (t_k - y_k) * f'(y_{\text{in}_k})$ 
    Find exchange value for weights between output-hidden layers
     $\Delta w_{jk} = \mu * \partial_k * z_j$ 
    For each hidden layer
      Calculate error
       $\partial_{\text{in}_j} = \sum \partial_k * w_{jk}$ 
    End for
    Find error value that effects input layer
     $\partial_j = \partial_{\text{in}_j} * f'(z_{\text{in}_j})$ 
    Find exchange value for weights between hidden-output layers
     $\Delta v_{ij} = \mu * \partial_j * x_i$ 
    Update all weights
     $w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}$ 
     $v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}$ 
  End for
Until termination criteria are met

```

Figure 3.9 Incremental backpropagation NN algorithm

3.3.2 Performance Calculation of the Models

There are two parameters that effect performance of NN models. Performance of model is inversely proportional to the *error_value* and *iteration number*. The less error value and the iteration number decreases, the more performance of the models increases.

Definition 17 Performance of the models is calculated as follows;

$$P_m = \frac{1}{IN * error_value}$$

where P_m is performance of the model, IN is iteration number and $error_value$ is result error value of the model.

3.4 Experimental Results

3.4.1 Model Construction

All experiments in this study are performed by using a tool, named Neural Network Modeller Tool, which was developed in Visual Studio 2008 using C# as programming language and run on a laptop with Intel Core2Duo 2.0 GHz CPU and 2GB of RAM under the Window Vista operating system.

After each training operation a specific model that includes all information about training, is created. An example of that model is illustrated in Figure 3.10. This model includes NN type (Backpropagation, SLP, MLP or SOM), total layers number, dimensions of network (neuron numbers of layers), iteration number and error value when iteration terminates, activation function type (1→Sigmoid, 2→Hyperbolic Tangent,

3→Gaussian, 4→Piecewise linear , 5→Identity , 6→Unit Step), gradient and learning rate values and weights and biases.

```

Backpropagation
Network Layer Number :4
Network Dimensions :2 4 2 1
Iteration Number :720
Error :0,00998562713982413
Function :1
Gradient :5
Learn Rate :0,1
WEIGHTS
1 1,1 :0,884300548463931
1 2,1 :0,0112247480317906
1 1,2 :-0,0842624762493482
1 2,2 :-0,259148489434602
1 1,3 :0,1545134836283
1 2,3 :-0,342639268456297
1 1,4 :0,314832222686472
1 2,4 :0,200922476303402
2 1,1 :0,825271709380552
2 2,1 :0,174779887163579
2 3,1 :0,0198953408931536
2 4,1 :-0,354901089339561
2 1,2 :0,308345816422373
2 2,2 :-0,463675651606788
2 3,2 :0,228017388462221
2 4,2 :0,64104074515338
3 1,1 :0,679056741580004
3 2,1 :-2,05713252370634
0
0
0,000534933136012789
5,45932629712011E-05

```

Figure 3.10 An example classifier model

The following notations are used in the proposed model:

I_n	Input Layer neuron number
O_n	Output Layer neuron number
HL	Hidden Layer number
HL_n	Hidden layers neuron numbers
AF	Activation Function
μ	Learning Rate
β	Gradient Value
Restart	Iteration number that training will restart after that

T_c	Termination Criteria
IN	Iteration number as termination criteria
min_error	Error value as termination criteria

3.4.2 *Description of the Dataset*

The tool that is developed for classification by using neural network, can be applied all datasets which include integer values. But to make sensitivity analysis of NN parameters three datasets for supervised algorithms, are used. Classification experiments are conducted on the real-world datasets from the UCI Machine Learning Repository (Asuncion & Newman, 2007). One of them is wine dataset that includes 178 patterns, 3 classes and 13 attributes. Other dataset is iris dataset. Iris dataset includes 150 patterns, 3 classes and 4 attributes. Beside these datasets a small dataset that called 'patterns dataset' and includes 2 attributes, 2 classes, to compare effects of attribute and pattern numbers. For SOM which is unsupervised NN, food dataset which includes patterns with 4 attributes is used.

3.4.3 *Iteration Number*

To show effect of iteration number on error value, 120 training operations are done for two datasets, wine and patterns datasets. Backpropagation NN is applied for that investigation. 60 training are done for wine dataset and 60 training operations are done for patterns dataset. 6 different INs (10, 20, 30, 40, 80 and 160) are applied as T_c and for each IN, 10 training operations are applied and then average of error values of these operations is taken. NN parameters that are used for training operation for wine dataset is shown in Table 3.1, for patterns dataset is shown in Table 3.2. Blue line represents error value for wine dataset and pink line is for patterns dataset. As it can be seen from the Figure 3.11, error value difference between two datasets is so much. Because wine dataset includes patterns nearly 6 times of patterns dataset. And there are 13 attributes in

wine, only two attributes in patterns dataset. It can be understood from this investigation, attribute number and pattern number effect error value, and also error value depends on dataset.

Other result of that observation is error value change amount between INs. It can be seen from the graphic that is shown in Figure 3.11, error value decrement is big for the first INs. And then decrement decreases. For example for the wine dataset, for the 10 INs, error value is 153.777, for the 20 INs error value is 119.037. Change amount is nearly 34. But change between 20 and 30 INs is only 9. This is the same for the second dataset, too. So we can say that the more IN increases, the less error value changes.

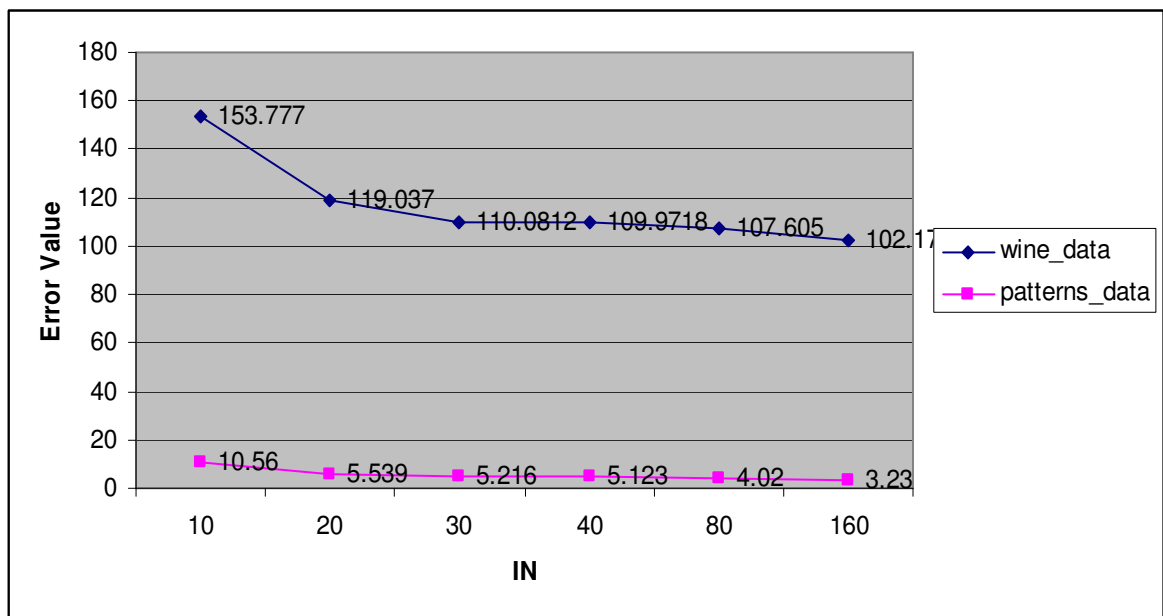


Figure 3.11 IN-Error_Value graphic

Table 3.1 Parameters of training for IN-Error Value Involvement examination for wine dataset

I_n	O_n	HL	HL_n	AF	μ	β	Restart	T_c
13	3	2	4,2	Sigmoid	0,01	6	-	IN

Table 3.2 Parameters of training for IN-Error Value Involvement examination for patterns dataset

I_n	O_n	HL	HL_n	AF	μ	β	Restart	T_c
2	1	2	4,2	Sigmoid	0,01	6	-	IN

3.4.4 Hidden Layers

To see effect of hidden layer numbers on error value, 100 training operations are applied using Backpropagation NN on patterns dataset. 50 training operations are applied with two layers, with NN parameters that are listed in Table 3.3 and 50 training are applied with three layers and with parameters that are listed in Table 3.4. As it can be seen in the Figure 3.12, for the simple problems, the more hidden layer numbers increases, the more IN is needed for minimum error. But with two hidden layer number, training gets snagged local minimum more frequent then three hidden layer NN.

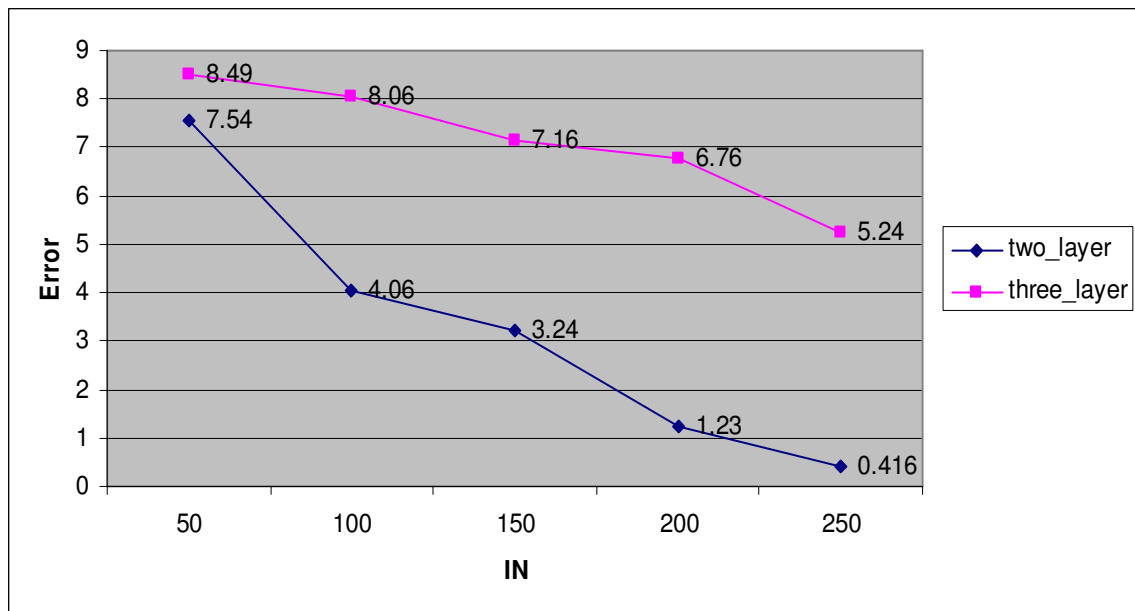


Figure 3.12 Hidden layers effect graphic

Table 3.3 Parameters of training for 2 hidden layers effect examination

I_n	O_n	HL	HL _n	AF	μ	β	Restart	T _c
2	1	2	4,2	Sigmoid	0.01	5	-	IN

Table 3.4 Parameters of training for 3 hidden layers effect examination

I_n	O_n	HL	HL _n	AF	μ	β	Restart	T _c
2	1	3	4,2,2	Sigmoid	0.01	5	-	IN

3.4.5 Learning Rate (μ)

Learning rate effects training time of network. Learning rate should be chosen carefully. 60 training operations are done by using Backpropagation NN to recognize importance of μ . Figure 3.13 shows results of these trainings. 30 trainings are done by using sigmoid function as activation function, and 30 trainings are done by using unit step activation function. Fixed NN parameters that are used during training operation are listed in Table 3.5. As it can be seen from graphic that is illustrated in Figure 3.13, the more learning rate increases, the more velocity of the network increases. For example when $\mu=0.01$, IN is 1509 to reach min_error (0.01), but when $\mu=0.04$ then IN=519 for the same min_error value. But learning rate prevents NN from local minimum, and also μ prevents from random search in search space. Another inference from that investigation is importance of activation functions. For the simple problems unit step is more suitable than another complex activation functions. For example for the $\mu=0.04$, NN that uses sigmoid applies 519 iterations to reach solution, but when unit step is used then this value is only 9. But for the complex problem unit step is not enough.

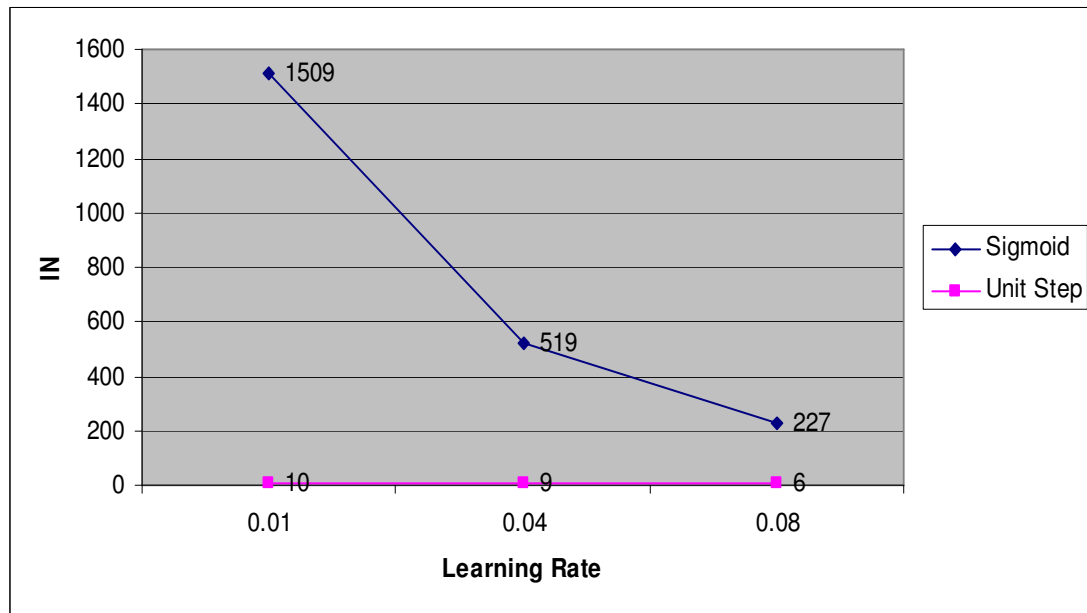


Figure 3.13 Learning Rate

Table 3.5 Parameters of training for learning rate examination

I_n	O_n	HL	HL_n	β	Restart	T_c	min_err
2	1	2	4,2	6	-	Minimum Error	0,01

3.4.6 Neural Network Types

In that study four neural network types are applied for classification. Backpropagation, Single Layer Perceptron and Multilayer Perceptron are supervised and SOM is unsupervised NN. So SOM classifies different kind of datasets from others. In that investigation Backpropagation, SLP and MLP are compared according to IN-Error Value couples. To achieve that investigation, 90 training operations are applied by using NN parameters that are listed in Table 3.6 and Table 3.7. IN is used as termination criterion. Three different INs (20, 40 and 80) are used. As it can be seen from the graphic that is illustrated in Figure 3.14, most successful NN type between three NN types is Backpropagation. For 20 INs error value for Backpropagation is 118, for MLP 120.3 and for SLP error value is 287.

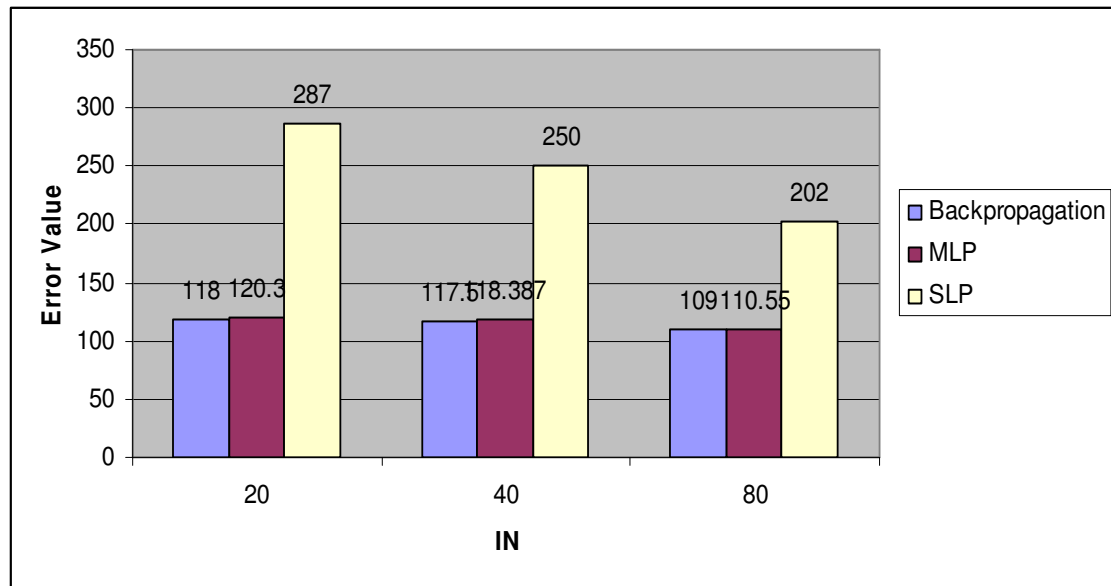


Figure 3.14 NN Types Comparison

Table 3.6 Parameters of training for NN Types Comparison examination for Backpropagation and MLP

I_n	O_n	HL	HL_n	β	AF	T_c
13	2	2	4,2	5	Sigmoid	IN

Table 3.7 Parameters of training for NN Types Comparison examination for SLP

I_n	O_n	HL	HL_n	β	AF	T_c
13	2	-	-	5	Sigmoid	IN

3.4.7 Incremental vs. Traditional Backpropagation NN

In this section, the performances of traditional Backpropagation NN and incremental Backpropagation NN are compared through the experiments based on the real-world dataset wine. The implements of both algorithms were run with various parameter settings and every classification problem has been solved 10 times where average minimum costs were calculated. Consequently, 60 different experiments were handled for this purpose.

In the first case, a number of instances were removed from the wine dataset and training operation was done and specific model is created for that dataset. After that, incremental NN is applied for just new patterns, and traditional NN is applied for whole dataset. Comparison of two models is shown with graphic that is shown in Figure 3.15. Fixed parameters that are used in this training operation are shown in Table 3.8. Three different numbers (450, 300 and 150) of instances were removed from the dataset. The less number of missing patterns decreases, the more difference of IN between traditional and incremental increases. For example for the 450 missing patterns, incremental NN has only 2024 IN but traditional NN has 3527 INs. Incremental NN provides high speed training.

Table 3.8 Parameters of training for Incremental-Traditional Backpropagation NN Comparison

I_n	O_n	HL	HL _n	β	AF	T_c	min_error
13	2	3	6,4,2	5	Sigmoid	min_error	0.1

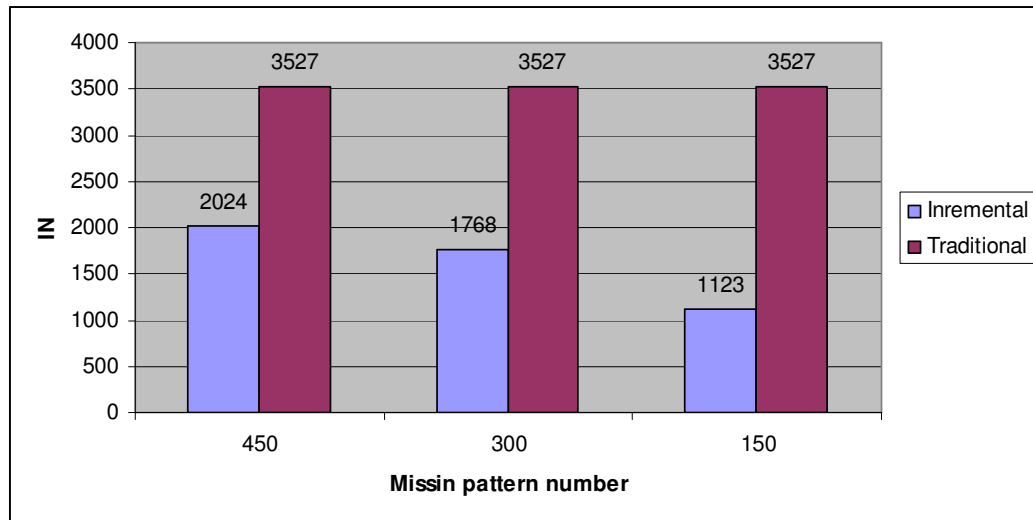


Figure 3.15 Incremental NN vs. Traditional NN

3.5 Interface

All operations are applied by using a tool that is developed for classification by using neural network algorithm and called “Neural Network Modeller”. This tool is developed in Visual Studio .Net 2008, using C# as programming language. Graphs are drawn by using Zed Graph.

All neural network parameters are entered as inputs. This provides user control and models that are created with different neural network parameters, comparisons.

This tool provides classification with Backpropagation NN, SLP, MLP and SOM, testing, model comparison and Incremental NN.

3.5.1 *Backpropagation*

This function of the tool provides classification with Backpropagation NN. This page that is illustrated in Figure 3.16 includes 6 main parts. The first part is for dataset. Dataset file name is entered manually or by using search button. After that, splitter that separates attributes in the dataset is selected (comma, space or full stop).

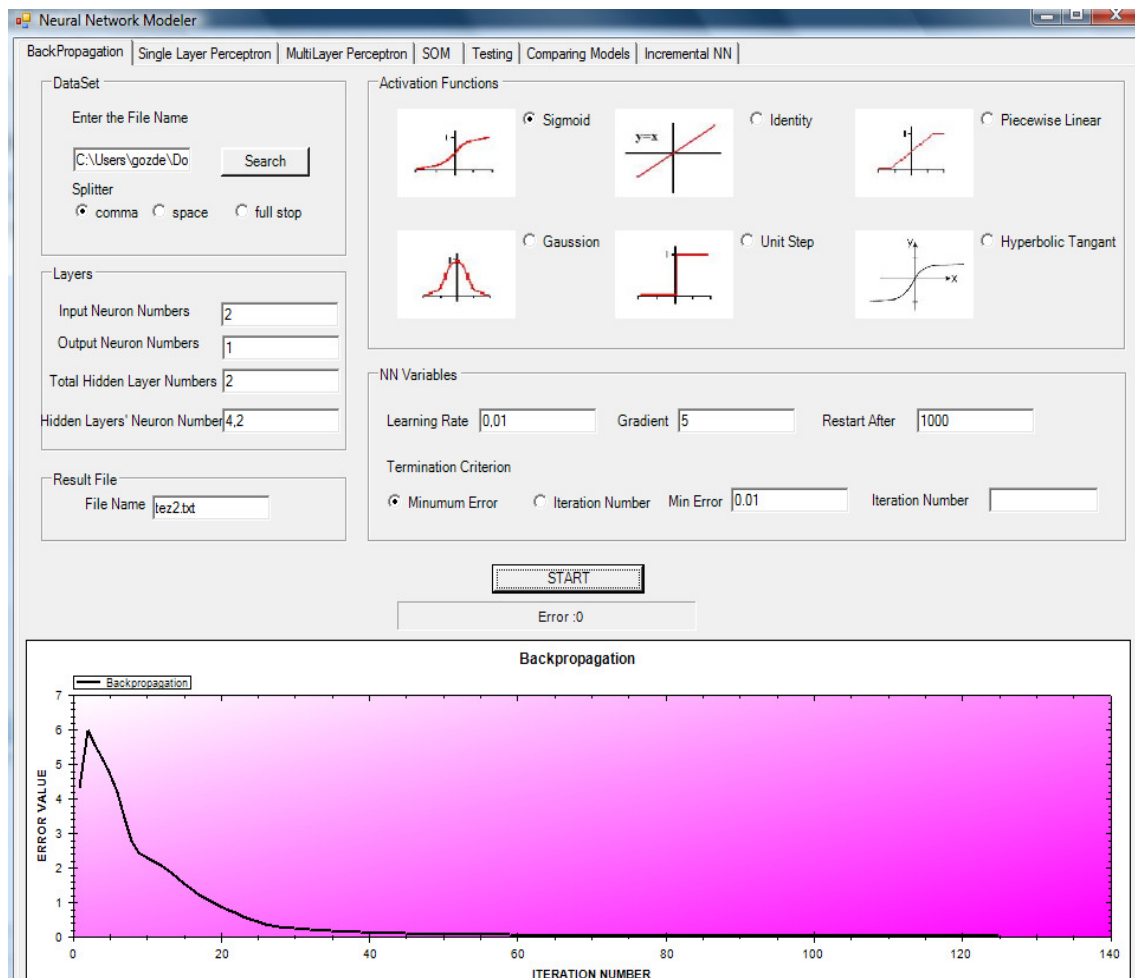


Figure 3.16 Classification with backpropagation

Second part is for layers of neural network. Input layer neuron number and output layer neuron number are entered. For example for that training operation that is shown in Figure 3.16 input layer neuron number is 2 and output layer neuron number is 1. After that, hidden layer number that stands for how many hidden layers number will be in the NN, is entered. For example it is 2 for that example. After that, neuron numbers of hidden layers are entered in sequence. For example it is '4, 2' in the Figure 3.16. That means, first hidden layer includes 4 neurons and second one includes 2 neurons. There should be comma between numbers.

Third part is for result model file name. Model of that training operation is created with that name. All information is saved in that file.

Fourth part is for activation function selection. There 6 alternatives. These are; Sigmoid, Identify, Piecewise linear, Gaussian, Unit step and Hyperbolic tangent activation function.

Other part of the page is for NN parameters. Learning rate and gradient are used in activation functions. Restart after is used to prevent local minimum. If error value is not smaller than min_error after a number of iteration, that means there is a local minimum situation. So network is initialised in that situation. This number is determined by user, and value of restart after depends on problem. For example it is 1000 for that training. After that, termination criterion is selected. There are two alternatives; *Minimum Error* and *Iteration Number*. If minimum error is selected as termination criterion then, the user should enter min_error value. If IN is selected then, iteration number should be entered.

After 'Start' button is pressed, training starts and Error-Iteration Number graphic is drawn. And also progress bar shows error value detailed.

3.5.2 SLP

This property of the tool provides classification with single layer perceptron. All parts are the same with backpropagation page except layers of network part. Because there are only input and output layers as distinct from backpropagation in the SLP. So there is not any textbox to take information about hidden layers. Only neuron numbers of input and output layers are taken from users. Figure 3.17 shows SLP page in the tool.

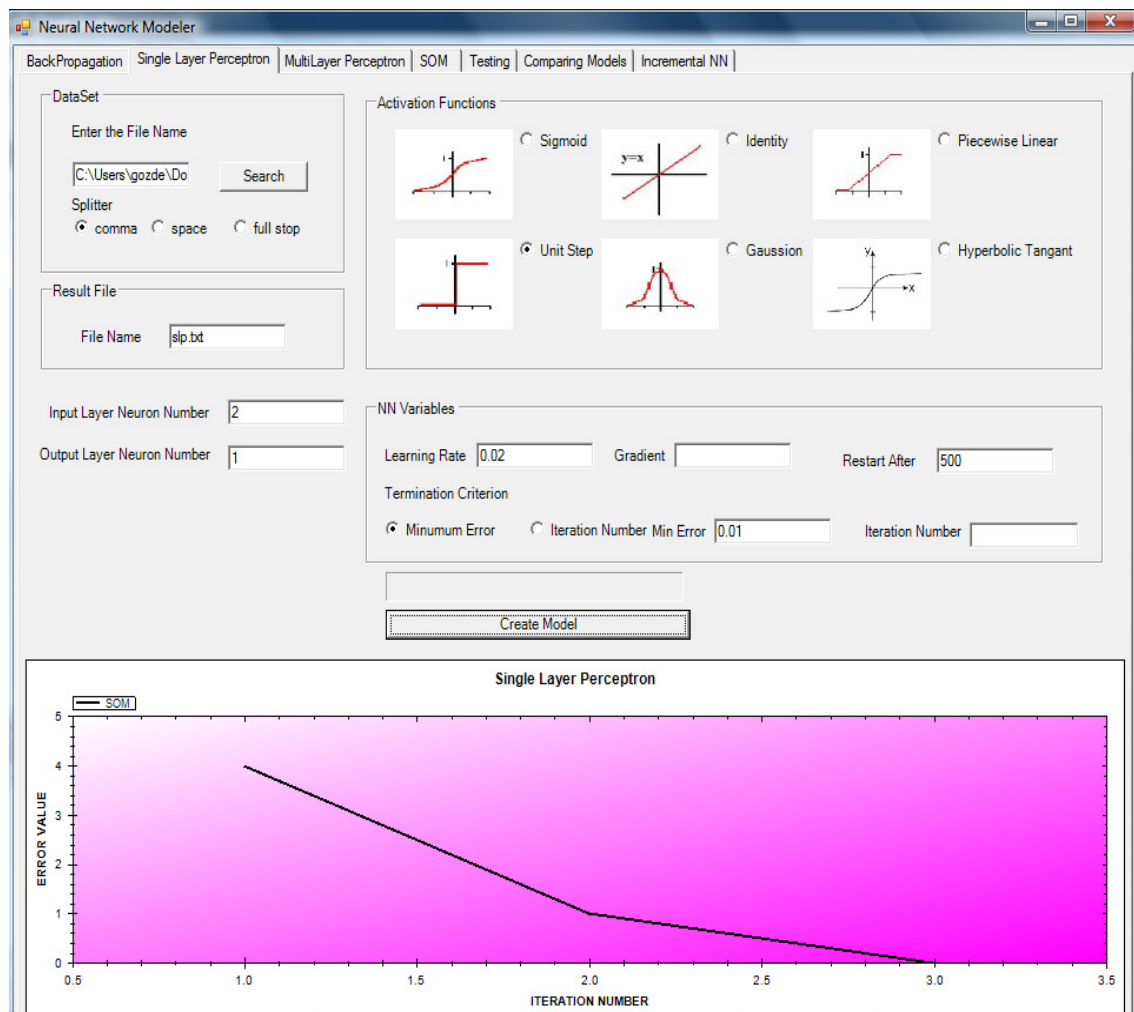


Figure 3.17 SLP

3.5.3 MLP

Appearance of this function of the tool is nearly same with backpropagation page. There is not any difference between two pages. But background of the pages is different because of algorithms.

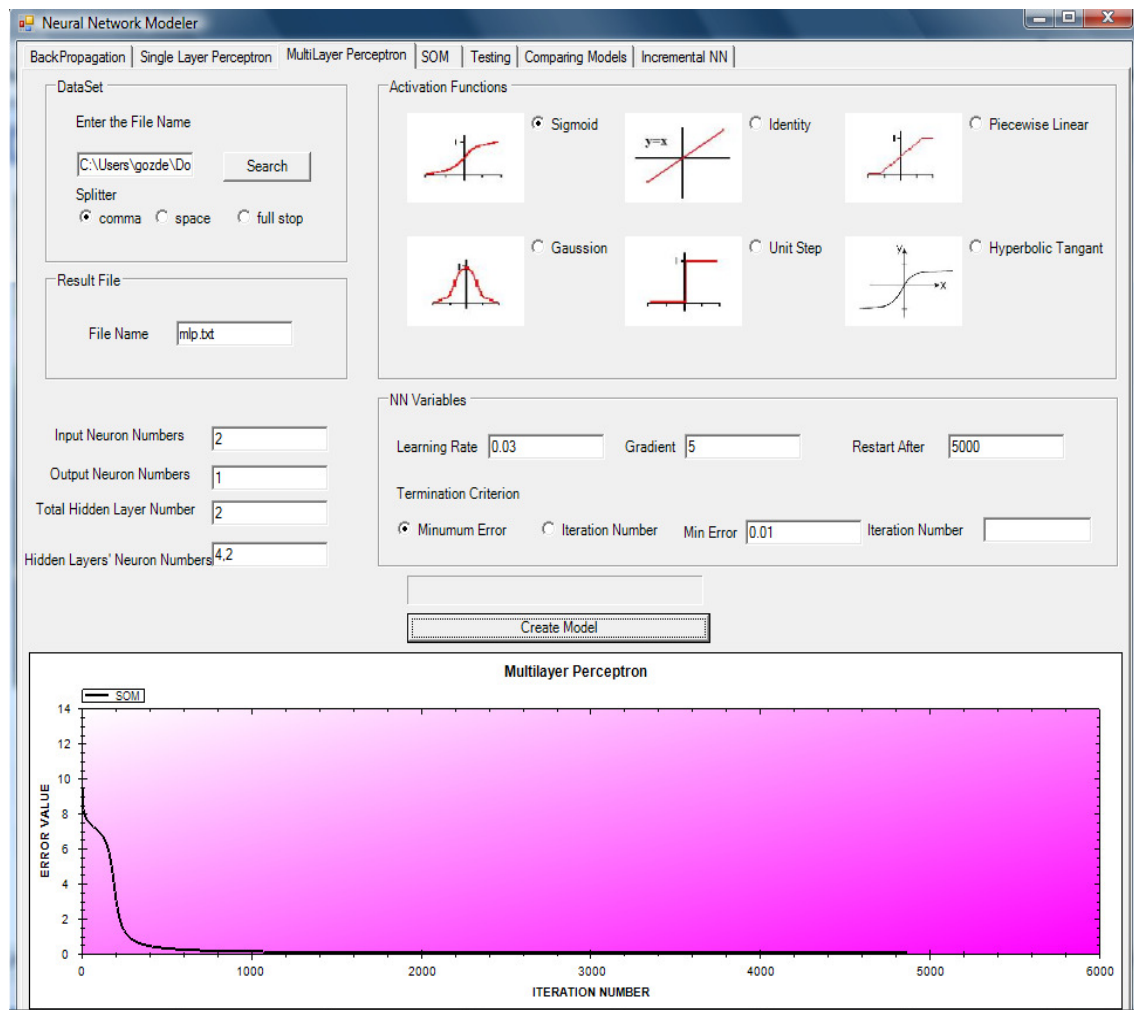


Figure 3.18 MLP

3.5.4 SOM

This is for classification with SOM algorithm. SOM page includes 4 parts. First part is for dataset file. Dataset file name is entered, and then splitter between attributes in the dataset is selected.

Second part is for SOM parameters. Input neuron numbers and length of output layer is entered manually. Minimum error value is entered by user. For example it is '0.00001' here. Model file name is written in 'result file name' textbox. There is a selection for

label info of the patterns. If patterns have labels, for example in that training that is shown in Figure 3.19, patterns have food names, these are labels of patterns.

After pressing 'start' button, classification operation starts. When training is finished then a graphic that shows position of the patterns, is drawn. In the graphic that is shown in Figure 3.19, patterns are disbanded in 10x10 coordinate system. Because length of output layer is 10 for that example. And x and y values and class information of patterns are listed in the datagridview.

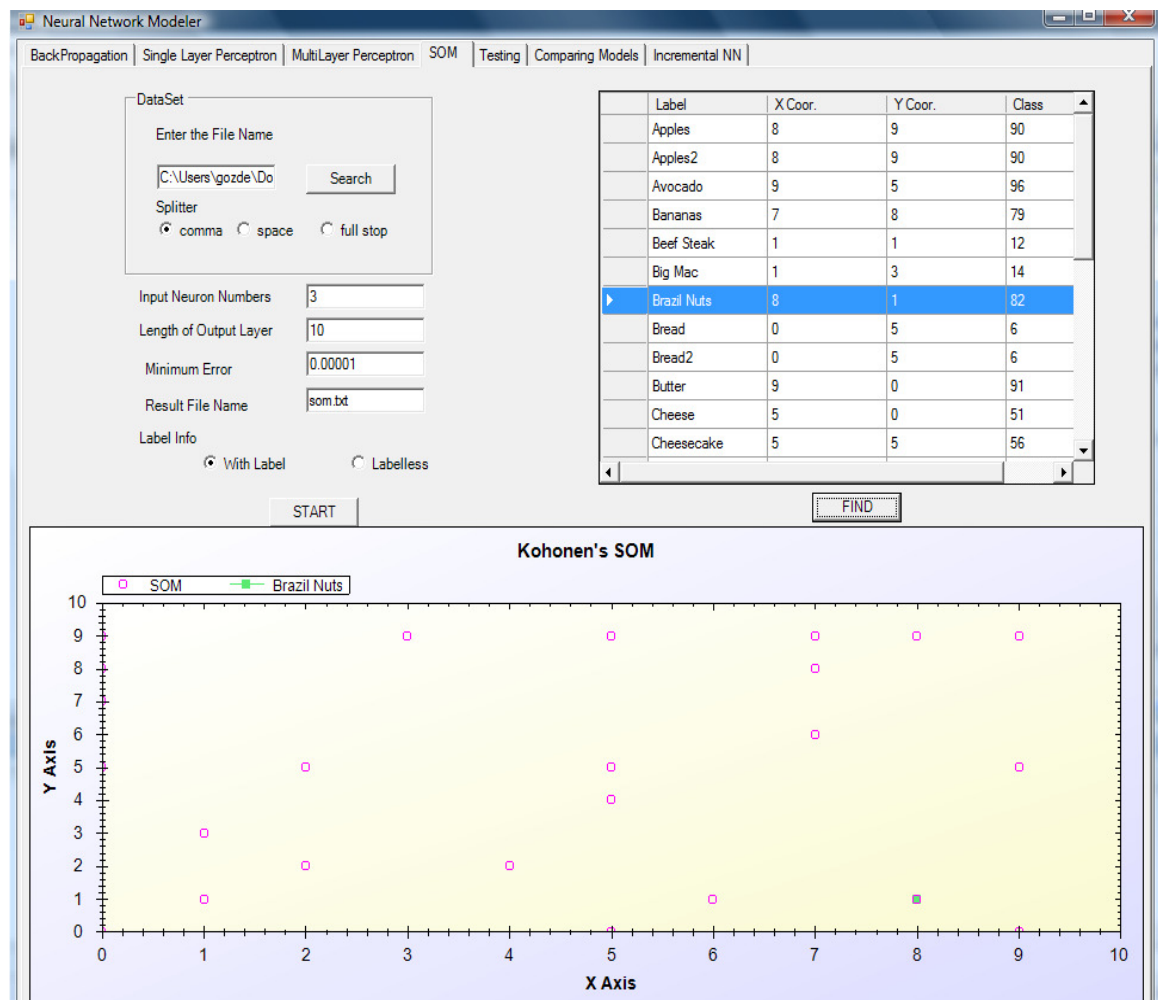


Figure 3.19 SOM

If the user wants to see the position of any patterns in the list in the datagridview, selects that patterns and then presses 'Find' button. For example, here in the figure, 'Brazil Nuts' is selected and 'Find' button is pressed, so (8,1) point is shown with green colour.

3.5.5 Testing

This page is to classify patterns that we don't know class value of them and validation test of the models. Figure 3.20 shows the interface. There are three main functions in the interface. First one is for classification of models that are created for backpropagation, SLP or MLP. Model file name is entered and then test file name that includes patterns that are wanted to classify, is entered. And then splitter that stands between attributes in the test file is selected. After that, output file name that class values of the patterns are written in sequence is entered manually. By pressing 'Test' button, model find class values of the patterns and writes results in the output file. The same operation is applied for the SOM models. If model is created for SOM, then second part of the page is used to classify.

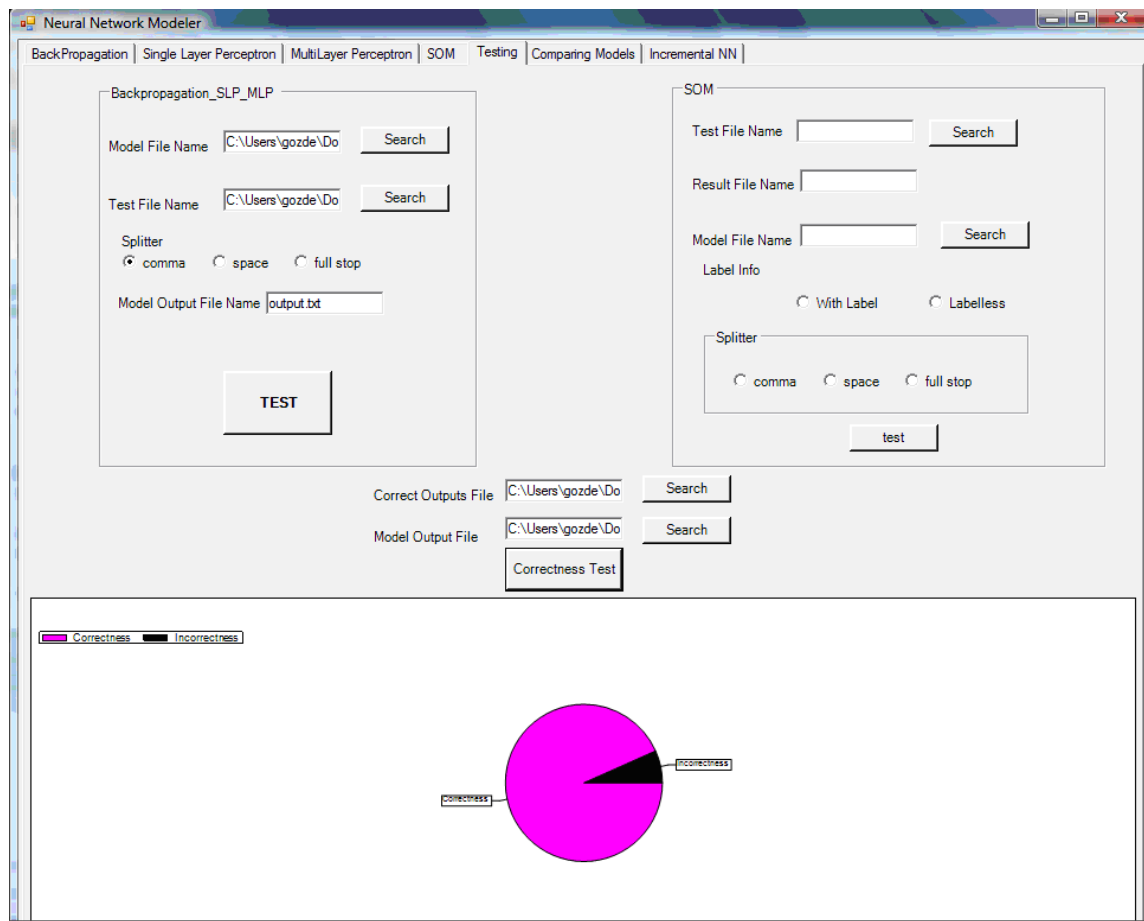


Figure 3.20 Testing

Third part of the page is for correctness testing. If class values of the patterns are known, then file name that includes correct class values, is entered manually or by 'Search' button. And then model output file name that is created by using test part of the page, is entered. By pressing 'correctness test' button, class values are compared, and result of the correctness test is shown by using pie chart that is drawn with Zed graph.

3.5.6 *Comparing Models*

This function of the model is to compare performance of the models. To compare of the models formula in the Definition 17 is used.

Firstly models are selected by using search button, and then by using add button these models are inserted in the datagridview list that is illustrated in Figure 3.21. For example there are two models in that model comparison operation. After that by pressing 'Compare' button, information (IN, Error, NN Type, Layers Dimensions and Activation Function) of models are written in the list and comparison result is shown in bar chart graph that is drawn by using Zed graph, as illustrated in Figure 3.21. For example for that comparison, performance of the second model is greater than first model. Because IN of the second model is 1225, and this is 2748 for the first model. The reason of the performance difference is shown in list in datagridview. For example the reason is activation function for this comparison. The result of that operation is, unit step function is more suitable than sigmoid function for current dataset.

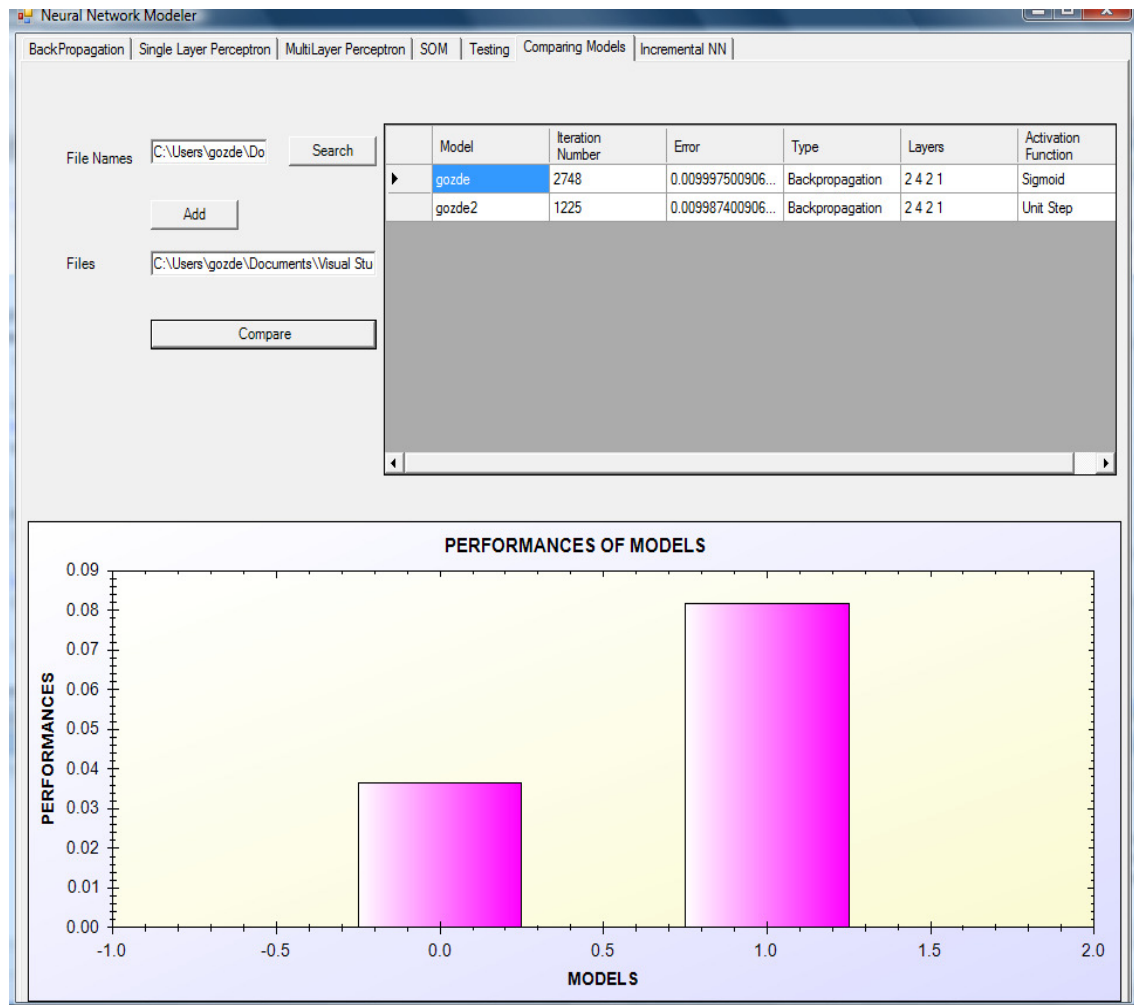


Figure 3.21 Comparing models

3.5.7 Incremental NN

When new patterns are added into a dataset which has NN model, dataset needs to classify again. In that situation all dataset with new patterns, should be trained. But with incremental NN, that neural network modeller tool provides, only new patterns are trained to classify all dataset. To achieve incremental NN, steps that are illustrated in Figure 3.10 are used. The interface that is shown in Figure 3.22 is 3rd and 4th steps of the incremental NN operation.

The screenshot shows the 'Neural Network Modeler' application window with the 'Incremental NN' tab selected. The interface includes the following elements:

- Navigation Tabs:** BackPropagation, Single Layer Perceptron, MultiLayer Perceptron, SOM, Testing, Comparing Models, Incremental NN.
- Model Selection:** Two radio buttons at the top: ☒ Backpropagation_SLP_MLP and ☐ SOM.
- File Selection:** Two text input fields for 'Model File Name' and 'New Dataset File Name', both containing the path 'C:\Users\gozde\Do'. Each field has a 'SEARCH' button to its right.
- Delimiter Selection:** A 'Splitter' section with three radio buttons: ☒ comma, ☐ space, and ☐ full stop.
- Termination Criteria:** Two radio buttons: ☒ min. error and ☐ iteraiton number (note the typo).
- Value Inputs:** Two text input fields: 'Minimum Error' with the value '0.01' and 'Iteration Number' which is empty.
- SOM Configuration:** A 'Label Info For SOM' section with two radio buttons: ☐ with Label and ☐ Labelless.
- Action:** A 'START' button at the bottom center.

Figure 3.22 Incremental NN

First of all NN types are decided. If NN type is one of Backpropagation, SLP or MLP “Backpropagation_SLP_MLP” radio button is checked, or if NN type is SOM, then “SOM” radio button is checked. After that operation, model file name that is created for previous dataset is selected by using “Search” button. And then file name that includes new patterns of dataset is entered. Splitter that stands between attributes in the new dataset file is selected. If minimum error is selected as termination criterion, and then minimum error value is entered, otherwise if IN is termination criterion then, IN is entered. If NN type is SOM then label situation is selected with label or labelless. By

pressing “Start” button training starts and when training operation is finished then all information of new model is saved in old model file, that is old model is refreshed.

CHAPTER FOUR

CONCLUSION

4.1 Conclusion for GA

Classification is a data mining task that assigns items in a dataset to target classes. Although genetic algorithms are less commonly used for classification in commercial data mining systems, this technique shows its strength in certain applications. In this study, an incremental GA-based classification is proposed for efficiently handling new transactions added into the dataset. The purpose of the algorithm is to decrease time needed for training to construct a new classifier with new dataset.

Numerous trials of incremental GA were run for the classification problem to determine the effects of various parameters on the performance of the algorithm. Because of the stochastic nature of the algorithm, several runs were made at each parameter setting to obtain an average. Parameters included crossover probability, mutation probability, with/without elitism, parent selection technique and population size. In addition, the performances of two different GA implementations (traditional and incremental) were compared according to the two different termination criteria (generation number and fitness value). All these experimental results show the reduction in average training time and improvement in score over the population and best individual score within the population at each generation. Incremental GA gives much better performance by adding rules into initial population before training operation when new transactions are added into the dataset.

A new formula is developed to determine performances of models. By using this formula, performances of models can be compared and so effects of GA parameters on classification operation can be comprehend.

To apply all of operations and analysis, a generic modeller, called “Genetic Algorithm Modeller” (GAM) is developed. This tool provides classification by using genetic algorithm and after classification operations. All datasets which include categorical data can be classified with GAM. GAM has ability to provide after classification operations. Beside classification, performance calculation of models, comparison performances of models, incremental GA, testing and validation test can be done by using GAM.

4.2 Conclusions for NN

In study an incremental NN-based classification is proposed for efficiently handling new transactions added into the dataset. The purpose of the algorithm is to decrease time needed for training to construct a new classifier with new dataset. To test effect of the algorithm several classification operations are applied for different datasets with different NN parameters. All these experimental results shows that Incremental NN gives much better performance by using weight values in the models instead of random generation before training operation when new patterns are added into the dataset.

A new formula is developed to determine performances of models. By using this formula, performances of models can be compared and so effects of NN parameters on classification operation can be comprehend.

Sensitivity analyses are applied to comprehend effects of NN parameters on performances of models. So optimum values of NN parameters are determined experimentally.

To apply all of operations and analysis, a generic modeller, called “Neural Network Modeller” (NNM) is developed like GAM. This tool provides classification by using neural network algorithm and after classification operations. All datasets which include numerical data can be classified with NNM, so this tool is generic. Classification can be

applied by using different neural network types; Backpropagation, SLP, MLP and SOM with NNM. NNM gives all controls to the user for training operation. To achieve this, values of all parameters that are used in activation functions, dimensions of layers and termination criteria are taken from user. Calculation performances of models, comparison of models, testing and correctness test can be done with NNM.

4.3 GA vs. NN for Incremental Classification

Incremental classification provides classification with less training time for GA and NN for datasets which are updated regularly. Model structure is developed to success incremental classification. Model structures are different for NN and GA. For NN; after initial classification operation that are applied for initial dataset, weights between neurons in the network structure are saved in model to use later. And for GA rules that represent each class in dataset are saved. For incremental NN, initial weights are initialised with weight values that are in model, not randomly. To achieve incremental GA, initial population is not created fully randomly. Rules in the model are added into initial population. For incremental NN, only new patterns are trained for classification, because we have weight values for first part of the dataset. So weights are updated by training only new patterns. But for GA, all dataset which include new patterns is trained. Because fitness function of GA needs all patterns in the dataset, not only new patterns. But initial population is not created fully randomly, rules which are found previous classification operation, are added into initial population. So training time decreases.

REFERENCES

- Asuncion, A., & Newman, D. (2007). UCI machine learning repository. Datasets "Nursery", "Wine", "Iris". *University of California, Irvine, Department of Information and Computer Sciences*. <http://archive.ics.uci.edu/ml/>
- Avci, E. (2009). A new intelligent diagnosis system for the heart valve diseases by using genetic-SVM classifier. *Expert System with Applications*, 36 (7), 10618-10626.
- Banerjee, A.K., Kiran, K., Murty, U.S.N., & Venkatesvarlu Ch. (2008). Classification and identification of mosquito species using artificial neural networks, *Computational Biology and Chemistry*, 32 (6), 442-447.
- Booker, L.B., Goldberg, D.E., & Holland, J.H. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, 40 (1-3), 235-282.
- Cacuci, D.G. (2003). *Sensitivity and Uncertainty Analysis: Theory*. Volume I, Chapman & Hall, 304 pages.
- Dehuri, S., Patnaik, S., Ghosh, A., & Mall, R. (2007). Application of elitist multi-objective genetic algorithm for classification rule generation. *Applied Soft Computing*, 8 (1), 477-487.
- Fan, Y-N., Tseng, T-L., Chern, C-C., & Huang, C-C. (2009). Rule induction based on an incremental rough set. *Expert Systems with Applications*, 36 (9), 11439-11450.
- Fan, Z., Chen Y., Ma, J., & Zhu, Y. (2007). Decision support for proposal grouping: A hybrid approach using knowledge rule and genetic algorithm. *Expert System with Applications*, 36 (2), 1004-1013.

- Freitas, A. (2003). A survey of evolutionary algorithms for data mining and knowledge discovery. *Advances in Evolutionary Computation*, Springer-Verlag, 819-845.
- Gen, M., & Cheng, R. (2000). *Genetic Algorithms and Engineering Optimization*. New York: John Wiley and Sons.
- Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Hancock, P. (1994). An empirical comparison of selection methods in evolutionary algorithms. *Lecture Notes In Computer Science*, 865, 80 - 94.
- Holland, J. (1975). *Adaptation in natural and artificial system*. Ann Arbor, University of Michigan Press.
- Ishibuchi, H., Nakashima, T., & Murata, T. (2001). Three-objective genetics-based machine learning for linguistic rule extraction. *Information Sciences*, 136 (1-4), 109-133.
- Kim, J-H. (2009). Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics & Data Analysis*, 53 (11), 3735-3745.
- Kim, M. W., & Ryu, J. W. (2005). Optimized fuzzy classification using genetic algorithm. *Lecture Notes in Computer Science*, 3613/2005, 392-401.
- Kwong, C.K., Chang, K.Y., & Tsim, Y.C. (2008). A genetic algorithm based knowledge discovery system for the design of fluid dispensing processes for electronic packaging. *Expert Systems with Applications*, 36 (2), 3829-3838.

- Lin, C-W., Hong, T-P., & Lu W-H. (2009). The Pre-FUFP algorithm for incremental mining. *Expert Systems with Applications*, 36 (5), 9498–9505.
- Lühr, S., & Lazarescu, M. (2009). Incremental clustering of dynamic data streams using connectivity based representative points. *Data & Knowledge Engineering*, 68 (1), 1-27.
- Manevitz, L., & Yousef M. (2007). One-class document classification via Neural Networks. *Neurocomputing*, 70 (7-9) 1466-1481
- Mazurovski, M.A., Habas, P.A., Zurada, J.M., Lo, J.Y., Baker, J.A., & Tourassi, G.D. (2008). Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural Networks*, 21 (2-3) 427-436
- Molnár, L., Keserű, G.M., Papp, A., Lőrincz, Z., Ambrus, G., & Darvas, F. (2006). A neural network based classification scheme for cytotoxicity predictions: Validation on 30,000 compounds. *Bioorganic & Medicinal Chemistry Letters*, 16 (4) 1037-1039
- Nath, A., Rahman, S., & Salah, A. (2005). An enhancement of k-nearest neighbor classification using genetic algorithm. Proceedings of the Midwest Instruction and Computing Symposium, Eau Claire, Wisconsin, USA.
- Rechenberg, I. (1971). Evolutionsstrategie - *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis. Technical University of Berlin.
- Saltelli, A. (2008). *Sensitivity Analysis*. WileyBlackwell, 492 pages.

- Shapiro, J. (2001). Genetic algorithms in machine learning. *Lecture Notes in Computer Science*. 2049/2001, 146-168.
- Übeyli, E.D. (2009). Combined neural network model employing wavelet coefficients for EEG signals classification. *Digital Signal Processing*, 19 (2) 297-308
- Yılmaz, T., Yıldırım, Y., & Yazıcı, A. (2007). Ontology-supported object and event extraction with a genetic algorithms approach for object classification. *Proceedings of the 6th ACM international conference on Image and video retrieval (In CIVR'07)*, New York, NY, USA, 202-209.
- Yu, B., & Zhu, D. (2007). Combining neural networks and semantic feature space for a-mail classification. *Knowledge-Based Systems*, 22 (5), 376-381
- Yuen, C.W.M., Wong, W.K., Qian, S.Q., Chan, L.K., & Fung, E.H.K. (2009). A hybrid model using genetic algorithm and neural network for classifying garment defects. *Expert System with Applications*, 36 (2), 2037-2047.
- ZedGraph. (n.d.). Retrieved March 3, 2009, from
http://zedgraph.org/wiki/index.php?title=Main_Page
- Zhu, F., & Guan, S. (2004). Feature selection for modular GA-based classification. *Applied Soft Computing*, 4 (4), 381-393.