ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF ENGINEEERING
M. Eng.

BY
CADENA, Carlos

EVOLUTIONARY FEATURE CREATION FOR ENSEMBLES

MONTREAL, MARCH THE 4TH, 2008

THIS THESIS WAS EVALUATED

BY THE FOLLOWING BORAD OF EXAMINERS

M. Robert Sabourin, Thesis Supervisor
Department of automatic manufacturing engineering at École de technologie supérieure

M. Patrick Maupin, Thesis Co-supervisor (external)
Defence Research and Development Canada (DRDC Valcartier)

Jacques-André Landry, President of the Board of Examiners
Department of automatic manufacturing engineering at École de technologie supérieure

M. Christian Gagné, External examiner
MacDonald, Dettwiler and Associates Ltd.

THIS THESIS WAS PRESENTED AND DEFENDED

BEFORE A BOARD OF EXAMINERS AND PUBLIC

FEBRUARY 21, 2008

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGMENTS

I would like to thank my thesis supervisor, Prof. Robert Sabourin, for giving me the opportunity of working within his group, for his constant guidance and for his patience all this time. When we were stuck, he always had a new perspective that enables us to go on.

Also, I would like to thank Patrick Maupin, thesis co-supervisor. His help throughout, gave cohesion to the thesis in different points. Part of this project was financed by DRDC-Valcartier under the contact W7701-2-4425/001/QCA.

I would like to thank two people related to Open BEAGLE: Christian Gagné and Matthew Walker. They helped me a lot to understand it and fix some issues during the development phase. In addition, Christian Gagné accepted to be part of the Board of Examiners.

My thanks go as well to Jacques-André Landry, who accepted to be the President of the Board of Examiners.

Special thanks go to Eulanda M. de Santos, who was always willing to help me and to answer the 1001 questions that I formulated. In the end, she was involved in this thesis more than she would have thought.

My wife Olga and my two children Alejandro and Sebastian were merely alone all through this time (I was virtually with them but my mind was far in the ensembles). Even though, Olga surrounded me with love and support (emotional and financial) and my children patiently supported not having their daddy at home so many evenings and weekends. I dedicate to them what this thesis means and all good things that will come as a result. Thank you for loving me.

People from Livia laboratory helped me a lot all the way. I would like to mention Vincent Dore, Marcelo Kapp, Albert Ko, Dominique Rivard, Jean Francois Connolly, Eric Thibodeau, Paulo Cavalin, Paulo Radtke, Luis Da Costa, Mathias Adankon and all others.

EVOLUTIONARY FEATURE CREATION FOR ENSEMBLES

CADENA, Carlos

ABSTRACT

Evolutionary feature creation for ensembles is about the generation of new attributes useful to build classifiers and ensembles of classifiers (EoC), based on evolutionary algorithms. The new attributes consist in transformations applied to the original raw features into a different space with the same or smaller cardinality, so that the subsequent classification process is simpler to be executed and provide better results. The feature creation process is intended towards the generation of ensembles using the built classifiers.

Bot's method is based on Genetic Programming (GP) (Koza, 1992), which "builds the features" that define the classifier. GP is used because it has the ability to discover underlying data relationships and express them mathematically (Kishore *et al.,* 2000) establishing the structure and values of the solution (Guo *et al.,* 2005). As the evolution progresses, GP discards the raw features that are not useful to solve the problem. Thus, by applying genetic programming we are doing feature construction and a sort of feature selection at the same time.

The method to generate the classifiers is based on a method proposed in (Bot, 2001) and it is called here Bot's method because of its author. Bot's method uses GP and consists in creating one feature at a time and guiding the evolution of new evolved features with the aid of the improvement in recognition rate of the proposed new feature in conjunction with the already evolved features. Bot's method improves performance with each new evolved feature by adding diversity and eluding the over-fitting phenomenon. We have improved Bot's method in two ways: adding a global validation procedure to control the over-fitting and setting it with the island method.

The classifiers created by building the features based on GP are called *evolved classifiers* and they represent the elements of the ensembles to be generated. We choose random subspaces method (Ho, 1998b) to generate ensembles and we have proposed two strategies to create EoC. In the first one, we combine the votes from each evolved classifier feature by feature. The performance obtained is slightly better than an ensemble of raw random subspaces. What is more, the same performance level of an ensemble of raw random subspaces is attained after some features. As a result, we can build EoC to assure certain performance with the minimum number of evolved features. This reduces the complexity of the ensemble without reducing the performance. The second strategy proposed is to create the ensembles based on finding for each base classifier, the maximum number of evolved features before over-fitting the optimization data set. The base classifiers have then different number of evolved features but each one provides the best recognition rate controlling at maximum the over-fitting. Also,

in this case we built ensembles with better performance than the ensemble of raw random subspaces. Furthermore, our performance results with cardinality of 9 to 12 evolved classifiers are close to the best ensembles reported in (Tremblay, 2004) with cardinality in the order of 30 base classifiers.

UNE MÉTHODE ÉVOLUTIONNAIRE POUR LA CONSTRUCTION DE CARACTERISTIQUES ADAPTÉE AUX ENSEMBLES DE CLASSIFICATEURS

CADENA, Carlos

RÉSUMÉ

La construction de caractéristiques par techniques évolutives pour la génération d'ensembles consiste en la génération de nouveaux attributs utiles pour la mise en œuvre de classificateurs (ou ses ensembles de classificateurs (EoC)) plus performants. Les nouveaux attributs sont définis par des transformations appliquées aux caractéristiques initiales avec une projection dans un autre espace de représentation de cardinalité identique ou inférieure, de sorte que la classification est plus simple d'exécution et peut fournir de meilleurs résultats. La construction des caractéristiques est généralement destinée à la génération d'ensembles qui utilisant les classificateurs déjà construits.

La méthode pour produire les classificateurs est basée sur la procédure proposée par (Bot, 2001) et appelée ici la méthode de Bot. La méthode utilise la programmation génétique (Koza, 1992) (PG), car elle a la capacité de découvrir des relations sous-jacentes dans les données et de les exprimer mathématiquement (Kishore *et al.*, 2000). La méthode de Bot consiste à créer une caractéristique à la fois et d'orienter l'évolution des nouvelles caractéristiques évoluées avec le taux de reconnaissance de la nouvelle caractéristique proposée en collaboration avec les caractéristiques déjà évoluées. Cette méthode améliore les performances lors de chaque nouvelle caractéristique évoluée en ajoutant de la diversité et en évitant l'impact négatif du surapprentissage. Nous proposons deux stratégies pour améliorer la méthode de Bot: (1) l'utilisation de la validation globale (Radtke *et al.*, 2006) qui permet de raffiner le critère d'arrêt et de contrôler le surapprentissage et (2) la mise en œuvre de la version parallèle de l'algorithme de Bot en utilisant la méthode des îles.

Les classificateurs créés par la construction des caractéristiques basée sur des évolutions de PG sont appelés *classificateurs évolués* et ils représentent les éléments des ensembles. Nous avons utilisé la méthode des sous-espaces aléatoires (Ho, 1998b) pour générer des ensembles et nous avons proposé deux stratégies pour créer ces ensembles. Dans la première stratégie, nous combinons les votes de chaque classificateur évolué, caractéristique par caractéristique. La performance obtenue est légèrement supérieure à un ensemble de sous-espaces aléatoires non évolués. En plus, le même niveau de performance qu'un ensemble de sous-espaces aléatoires non évolués est obtenu au bout de quelques caractéristiques. Par conséquent, nous pouvons construire des ensembles de classificateurs qui assurent un niveau satisfaisant de performance avec un nombre minimal de caractéristiques évoluées. Cela réduit la complexité de l'ensemble, sans réduire la performance. La deuxième stratégie proposée pour créer les ensembles est basée sur la recherche du nombre de caractéristiques évoluées pour chaque classificateur, de façon à contrôler le surapprentissage. Les classificateurs évolués ont donc

un nombre différent de caractéristiques, mais chacun offre le meilleur taux de reconnaissance en contrôlant le surapprentissage. Dans ce cas, nous avons construit des ensembles avec des performances supérieures à l'ensemble des sous-espaces aléatoire non évoluées. De plus, les résultats expérimentaux obtenus avec des ensembles de cardinalité variant de 9 à 12, sont proches de la meilleure performance obtenue pour les ensembles rapportés par (Tremblay, 2004), avec une cardinalité d'environ 30 classificateurs de base.

UNE MÉTHODE ÉVOLUTIONNAIRE POUR LA CONSTRUCTION DE CARACTERISTIQUES ADAPTÉE AUX ENSEMBLES DE CLASSIFICATEURS

CADENA, Carlos

SYNTHÈSE

La construction de caractéristiques par techniques évolutives pour la génération d'ensembles consiste en la génération de nouveaux attributs utiles pour la mise en œuvre de classificateurs (ou ses ensembles de classificateurs (EoC, de l'anglais *ensemble of classifiers*)) plus performants. Les nouveaux attributs sont définis par des transformations appliquées aux caractéristiques initiales avec une projection dans un autre espace de représentation de cardinalité identique ou inférieure, de sorte que la classification est plus simple d'exécution et peut fournir de meilleurs résultats. La construction des caractéristiques est généralement destinée à la génération d'ensembles utilisant les classificateurs déjà construits.

La classification est un problème important en reconnaissance de formes. Les objets d'un problème de la reconnaissance des formes sont modélisés grâce à une base de données. Par conséquent, l'objet d'un système de classification en reconnaissance de formes est d'assigner une catégorie ou classe aux échantillons de la base de données. Il n'y a pas de méthode de classification qui soit applicable à tous les types de problèmes de reconnaissance. Le degré de difficulté d'un problème de classification dépend de la variabilité des caractéristiques pour les objets dans la même catégorie relative à la différence entre les objets de différentes catégories (Duda *et al.,* 2001). Lorsque la complexité du problème augmente, le vecteur de caractéristiques qui décrit la base de données devient de plus en plus complexe et la difficulté de la tâche de classification est également augmentée. Pour tenter de résoudre ces problèmes, les chercheurs essaient de trouver un sous-ensemble de caractéristiques qui permet d'atteindre des résultats similaires, mais avec une réduction de la quantité de ressources et de temps nécessaires. Nous cherchons donc un sous-ensemble de caractéristiques qui permet de différencier les catégories. Une démarche complémentaire consiste à créer les éléments d'un ensemble de départ, de sorte que les classificateurs montrent collectivement une meilleure performance que le meilleur classificateur individuel.

Le deuxième chapitre décrit comment les techniques évolutives, en particulier les algorithmes génétiques (GA) et la programmation génétique (PG), peuvent être appliquées au problème de la sélection et de la construction de caractéristiques. Nous montrons que la programmation génétique est très utile pour s'attaquer au problème de la création des caractéristiques, dans le but de générer des classificateurs. La PG est utilisée car elle a la capacité de découvrir des relations sous-jacentes dans les données et de les exprimer mathématiquement (Kishore *et al.*, 2000), puis de découvrir la structure et des valeurs de la solution (Guo *et al.,* 2005). Au fur et a mesure que l'évolution progresse, la PG rejette les caractéristiques qui ne sont pas utiles pour résoudre le problème. Ainsi, en appliquant la PG,

nous faisons à la fois la construction de nouvelles caractéristiques et une sélection des caractéristiques de départ. Les classificateurs produits seront les éléments constitutifs des EoC, ce qui fait l'objet du troisième chapitre.

Le troisième chapitre traite des concepts de base des EoC. Il explique différentes techniques pour générer les ensembles comme la manipulation des exemples d'apprentissage, des caractéristiques d'entrée ou des sorties. Nous présentons aussi différentes façons de combiner les classificateurs de l'ensemble et l'application des techniques de sélection et de construction des caractéristiques à la sélection et la construction des ensembles. Les classificateurs créés par la construction des caractéristiques basée sur des évolutions de PG sont appelés *classificateurs évolués* et ils représentent les éléments des ensembles de classificateurs. Les EoC peuvent obtenir des résultats similaires, et parfois meilleurs que les performances d'un classificateur unique et complexe. Nous avons choisi une approche particulièrement intéressante pour créer des ensembles, basée sur la génération de sous-ensembles de caractéristiques à partir d'un ensemble de départ (Ho, 1998b). Cette méthode est appelée sous-espaces aléatoires (RS de l'anglais *random subspaces*). Dans notre cas, les classificateurs générés par RS sont évolués et ensuite combinés pour produire l'ensemble final. Bien que l'évolution des classificateurs par RS apporte de nouvelles connaissances sur le problème, chaque classificateur évolué n'a encore qu'une représentation partielle du problème de reconnaissance. Leur combinaison dans un ensemble permet une augmentation des performances par rapport à chaque classificateur individuel et aux ensembles de sous-espaces aléatoires non évolués.

La méthode utilisée pour produire les classificateurs est basée sur la procédure proposée par (Bot, 2001) et appelée ici la méthode de Bot. Cette méthode, qui est le sujet du chapitre 4, consiste à créer une caractéristique à la fois et d'orienter l'évolution des nouvelles caractéristiques évoluées avec le taux de reconnaissance de la nouvelle caractéristique proposée en collaboration avec les caractéristiques déjà évoluées. Cette méthode produit des solutions avec un petit nombre de caractéristiques évoluées et des taux de reconnaissance acceptables, en ne prenant que quelques générations pour l'évolution. La méthode de Bot atteint de bons résultats principalement pour les raisons suivantes: la population est réinitialisée, puisque l'évolution est lancée à nouveau pour chaque nouvelle caractéristique évoluée, cela augmente la diversité de la population. Cette fonctionnalité empêche que l'évolution soit piégée dans un maximum local. En outre, l'évolution est faite pour un certain nombre de générations, ce qui permet de minimiser l'impact du surapprentissage. Par conséquent, la méthode de Bot améliore les performances lors de chaque nouvelle caractéristique évoluée en ajoutant de la diversité et en évitant l'impact négatif du surapprentissage. Nous avons testé la méthode avec les mêmes bases de données utilisées dans (Bot, 2001) et nous avons obtenu des résultats comparables à ceux rapportés par l'auteur.

Le chapitre 5 présente une analyse détaillée de la méthode de Bot et propose deux stratégies pour l'améliorer: l'utilisation de la validation globale (Radtke *et al.*, 2006) qui permet l'amélioration du critère d'arrêt de la méthode de Bot et la mise en ouvre de la version parallèle de la méthode. Le critère d'arrêt utilisé dans la méthode de Bot détermine le moment

d'arrêter l'ajout de nouvelles caractéristiques évoluées en se basant sur l'augmentation des taux de reconnaissance obtenus lors de l'ajout d'une nouvelle caractéristique. Si l'augmentation au cours de la phase d'apprentissage est inférieure à un seuil, cette dernière caractéristique est rejetée et le processus d'évolution est arrêté. Le seuil est calculé à partir de l'augmentation relative de la performance dans la base d'optimisation au moment où la validation est maximale. Ce critère est comparable à un mélange entre la validation à la fin de l'optimisation et le rejet anticipé. On dit validation à la fin de l'optimisation, car la comparaison est faite à la fin de l'évolution pour chaque caractéristique et rejet anticipé, car la dernière caractéristique évoluée est rejetée si l'augmentation du taux de reconnaissance dans l'optimisation n'est pas plus grande que le seuil. Ce critère est lourd et il ne considère pas ce qui se produit au cours de l'évolution pour créer une caractéristique. Nous avons appliqué la validation globale (Radtke *et al.,* 2006), qui assure le pouvoir de généralisation de chaque solution (individus de la population) en testant les solutions avec un ensemble de données de validation pour chaque génération et chaque caractéristique évoluée. En conséquence, nous sommes en mesure d'identifier le meilleur point d'arrêt parce qu'aucune nouvelle amélioration n'est obtenue en validation. Nous avons testé avec succès la méthode de validation globale avec certains des ensembles de données utilisés dans (Bot, 2001). Nous avons remarqué que le nombre moyen de caractéristiques évoluées est plus grand que dans la méthode originale de Bot. Une analyse détaillée des deux algorithmes est présentée en fin de chapitre.

La deuxième amélioration présenté au chapitre 5, est basée sur la méthode des îles (Cantú-Paz et Goldberg, 2001) pour paralléliser l'algorithme de construction de Bot. Cette méthode consiste a faire évoluer des sous-groupes de populations isolées qui font parfois l'échange d'individus dans un processus de migration (Lin *et al.*., 1994; Alba et Troya, 1999; Fernández *et al.*, 2003; Gagne *et al.*, 2003). Comme les sous-groupes de populations explorent différentes régions de l'espace de recherche, nous obtenons différentes solutions au problème posé. L'avantage est que nous pouvons utiliser ces solutions de différentes manières: en sélectionnant la meilleure solution entre toutes, en générant un ensemble composé des meilleures solutions et en utilisant toutes les solutions différentes à la fois, au lieu d'une solution unique. Ces trois possibilités offrent des solutions adaptées pour la définition des ensembles de classificateurs.

Nous avons proposé deux méthodes différentes pour créer les ensembles de classificateurs dans le sixième chapitre. Dans la première approche, nous avons choisi quelques classificateurs à partir d'une composition de 100 sous-espaces aléatoires (RS). Les classificateurs ont différents niveaux de performance pour bien représenter l'ensemble original de 100 classificateurs. Chacun des classificateurs sélectionnés est reconstruit dans un espace évolué, puis les classificateurs évolués sont combinés à l'aide de vote à la majorité simple comme fonction de fusion, pour générer l'ensemble. Les résultats montrent que les performances sont améliorées avec l'ajout de chaque caractéristique. La performance obtenue est légèrement supérieure par rapport aux classificateurs individuels et par rapport à un ensemble de sous-espaces aléatoires non évolués. De plus, le même niveau de performance qu'un ensemble de sous-espaces aléatoires non évolués est obtenu au bout de quelques caractéristiques (cela varie de 15 à 20 caractéristiques). Par conséquent, nous pouvons

construire des ensembles de classificateurs qui assurent un niveau de performance acceptable avec un nombre minimal de caractéristiques évoluées. Cela réduit la complexité de l'ensemble, sans réduire la performance. La cardinalité de l'ensemble construit est augmentée en raison de trois fois le nombre des classificateurs évolués car trois îles sont utilisées pour générer les espaces de représentations reconstruits (évolués).

La deuxième méthode proposée pour créer les ensembles est basée sur la recherche du nombre de caractéristiques évoluées pour chaque classificateur, de façon à contrôler le surapprentissage. Les classificateurs évolués ont donc un nombre différent de caractéristiques, mais chacun offre le meilleur taux de reconnaissance en contrôlant le surapprentissage. Dans ce cas, nous avons construit des ensembles avec des performances supérieures à l'ensemble des sous-espaces aléatoire non évoluées. De plus, nos ensembles de résultats avec une cardinalité de 9 à 12, sont proches de la meilleure performance rapportée par (Tremblay, 2004) pour les ensembles de RS, avec une cardinalité d'environ 30 classificateurs de base.

Bien que la méthode de Bot génère une transformation de l'espace de caractéristiques initiales à un nombre de caractéristiques évoluées, il est important aussi d'analyser le degré d'utilisation des caractéristiques initiales dans les solutions évoluées. Une question se pose alors : combien de caractéristiques initiales sont utilisées et quelle est l'importance de chaque élément dans les solutions. L'importance relative de chaque caractéristique dépend des problèmes eux-mêmes, mais pour certaines bases de données, l'analyse souligne quelles sont les plus importantes caractéristiques pour la classification. L'analyse de l'utilisation des caractéristiques initiales est une première étape dans l'interprétation des arbres de décision où les solutions sont générées par la programmation génétique. Cette analyse, rarement présentée dans les travaux sur la programmation génétique, est une autre contribution de ce mémoire.

Nous pouvons résumer les contributions de ce mémoire: l'amélioration d'une méthode de génération de classificateurs basée sur la programmation génétique reposant sur deux stratégies, et une procédure de validation globale (Radke *et al.*, 2006) qui contrôle l'impact négatif du surapprentissage. Une deuxième contribution est l'utilisation de la méthode des îles pour la génération des classificateurs évolués applicable aux ensembles de classificateurs. Enfin, l'analyse de l'utilisation des caractéristiques permet, dans certains cas, de déterminer les caractéristiques initiales les plus discriminantes pour la classification.

TABLE OF CONTENTS

LIST OF TABLES

Page

LISTE DES FIGURES

Page

LIST OF ABBREVIATIONS

AQ-14	Inductive incremental learning program
BMI	Body-mass index
C4.5	Decision tree algorithm
DNA	Deoxyribonucleic acid
EF*f*	Evolved Feature *f*
EF(f)	Mathematical representation of Evolved feature *f*
EoBI	Ensemble of Best Island
EoC	Ensemble of classifiers
GA	Genetic Algorithm(s)
GEFS	Genetic Ensemble Feature Selection
GP	Genetic Programming
k-NN	*k*-nearest neighbour classifier
MDM	Minimal Distance to Means classifier
MLP	Multi-layer perceptron
MOMA	Multi-objective Memetic Algorithm
Nbr_Gen	Number of generations
NIST	National Institute of Standards and Technology
NIST SD19	NIST Scientific Database for hand-printed documents and character recognition
NSGA	Non-sorting Genetic Algorithm
NSGA-II	Fast and Elitist Non-sorting Genetic Algorithm
OB	Open BEAGLE

OO	Object oriented foundations
Open BEAGLE	Open Beagle Engine Advanced Genetic Learning Environment
PD	Projection Distance
RS	Random Subspace
SBS	Sequential Backward Selection
SBFS	Sequential Backward Floating Search
SFS	Sequential Forward Selection
SFFS	Sequential Forward Floating Search
STL	Standard Template Library
SMV	Simple Majority Vote
UCI	Machine Learning Repository (University of California, Irvine)
WMV	Weighted Majority Vote
XML	eXtensible Markup Language

LIST OF SYMBOLS

A_I	First subset of the training data set used to generate ensembles
B_I	Second subset of the training data set used to generate ensembles
$C(N, N_{RS})$	Number of possible combinations of N_{RS} elements out of N, $N_{RS} < N$
CHi	Chromosome of individual I in population P
Ci	Evolved Classifier i
C_I	Classifier generated by managing samples by their class labels
D	Threshold in Bot's stopping criterion
D0, D1, D2	Example of variable sin terminal set T
E	Combination of N_{RS} raw features
EoICi	Ensemble of Island (or intermediate) Classifier i
f	Current feature during evolution
F	Function set
f_a	average fitness of population P
f_i	fitness value of individual i in population P
f_{max_val}	feature number at maximal recognition rate in Validation
fit_{Fish}	Fitness measure based on Fisher criterion
$fit_{i,j}$	Fitness function for classes i and j based on Fisher criterion
g	Generation of the evolution
h	Hypothesis
h'	Alternate hypothesis
$\mathscr{H}$	Hypothesis space
H(P)	Phenotypic diversity of population P

I	Set of elements in Random Subspace RS of size N_{RS}
J(.)	Criterion to optimize when selecting the best subset of N_S features from N features. i.e.: probability of correct classification
K	Number of classes
k	Number of neighbours when using k-NN classifier
L	Number of times the training samples are split according to label class
M	Number of individuals in population P (size of the population)
	Natural number
N	Number of raw features
Nbr_Samples	Number total of samples in a data set (i.e.: training)
N_B	Number of samples randomly selected in Bagging
N_{RS}	Dimension of Random Subspace *RS*
N_S	Number of raw features in subset vector X_S ($N_S < N$)
P(g)	Population at generation g
P'(g)	Selected population from P(g) to apply genetic operations
p_i	Probability of individual i in population P
q	Number of two-class sets out of K classes
R	Number of classifiers in the ensemble
RR(Ci)	Recognition rate of evolved classifier Ci
RR(C_i-$_{OPT}$)	Recognition rate of evolved classifier Ci in Optimization
RR(C_i-$_{VAL}$)	Recognition rate of evolved classifier Ci in Validation
RR(RSi)	Recognition rate of raw random subspace RSi
RSi	*i-th* raw random subspace of the ensemble

S	Auxiliary Archive in Global validation
T	Terminal set
T_B	Training subsets used in bagging
V-best	Number of best solutions taken from each evolved classifier
W-best	Number of best solutions taken from each island and evolved classifier
X	Vector of N raw features
Xs	Vector of Ns features from X
x_i	Raw feature
λ	Empirical value set in the fitness based on Fisher criterion
β	Scale constant applied when fitness measure is based on Fisher criterion

CHAPTER 1

INTRODUCTION

Feature creation for ensembles is about the generation of new attributes useful to build classifiers and ensembles of classifiers. The new attributes consist in transformations applied to the original raw features into a different space, so that the subsequent classification is simpler to be executed and provides better results. The feature creation process is intended towards the generation of ensemble of classifiers (EoC).

Classification is a central problem in pattern recognition. The objects to be classified are described as a set of *raw features* also called the *feature vector*. The representation of samples as feature vectors engenders a *data set*. The objects of a pattern recognition problem are modelled by means of a data set. Therefore, the purpose of a classification system in pattern recognition is to assign a *category* or *class* to the samples of the data set. There is no a single classification method that can be applied to every kind of pattern recognition problem. Experience in this field shows that the most effective method for developing classifiers involves learning from example patterns (Duda *et al.*, 2001). This process is called *training* and it allows adjusting the parameters to differentiate between categories of data. The examples that are used during the learning process are a fraction of the data set, and are called the *training* set. A typical quantitative measure of the performance of a classifier counts the amount of patterns assigned to the right category or class and is called the *recognition rate*. Its counterpart is called the *error rate* and measures the percentage of examples miss-classified. The utility of any classification system resides in its ability to correctly classify unseen data, i.e. data that have not been used during the training process. Another portion of the data set, called the *test* data, is used to assess the real classifier performance. Even though the classifier can achieve a 100% performance during the learning by example process, its performance in classifying new unseen data can be less successful. In this case, the classifier has memorized the training data set instead of learning the "common

characteristics" of samples of the same category. This phenomenon is called *over-fitting* the training data set and constitutes one disadvantage of the learning from examples method. Many researches are still trying to adjust the complexity of the classifier to avoid or reduce the over-fitting without impacting too much the performance of the classifier.

The degree of difficulty of a classification problem depends on the variability in the raw feature values for objects in the same category relative to the difference between objects of different categories (Duda *et al.,* 2001). As the complexity of a problem increases, the feature vector that describes the data set becomes more and more complex, as well as the difficulty of the classification task, i.e. classification of images of handwritten digits or classification of cancer based on micro-array DNA data. In the first case, (Oliveira *et al.,* 2002, Oliveira, 2003b) created a description of the digit images as vectors of 132 raw features. In the latter, micro-array DNA data is described by thousand of characteristics (Hong and Cho, 2006). To manage these types of problems, researchers try to find out a *feature subset* that attains similar classification results than the whole set of features with a reduced dimensionality (fewer features). Different feature subset selection methods have been developed in the pattern recognition field to provide accurate classification (Devijver and Kittler, 1982; Jain and Zongker, 1997; Kudo and Sklansky, 2000). Some works (Ferri *et al.,* 1993; Kudo and Sklansky, 2000) analyze the applicability of *genetic algorithms* (see above) to feature subset selection and conclude that they are particularly suited for problems of large dimensionality, where most of the classical methods require prohibitive computation time (Ferri *et al.,* 1993).

Genetic algorithms (GA) are evolutionary techniques inspired by natural evolution. They work with a population of solutions instead of a single solution. Individuals can be coded as binary string of fixed length. For the feature subset selection problem, each position in the string means that this specific feature is selected (1) to compose the feature subset or not (0). From an initial randomly generated population, individuals (of the population) go through the processes of genetic operations over a sequence of generations. Individuals represent a possible solution to the problem and undergo an evolution process guided by the notion of individual's fitness (the performance of a possible solution, in this case, performance of the

resulting feature subset). "Better" individuals tend to survive meanwhile "bad" solutions tend to be eliminated (Pal and Wang, 1996). This process is repeated until a pre-defined number of generations is reached or until a degree of performance is achieved.

A different approach consists in "building the features" that define the classifier using Genetic Programming (GP) (Koza, 1992). GP is a variation of GA in which the evolving individuals are computer programs, instead of fixed length strings from a limited alphabet of symbols (Koza, 1992). It has the ability to discover underlying data relationships and express them mathematically (Kishore *et al.*, 2000). As the evolution progresses, GP discards the raw features that are not useful to solve the problem. Thus, by applying genetic programming we are doing feature construction and a sort of feature selection at the same time. Construction of functions can be done by using genetic programming because it is able to establish the structure and values of the solution while GA is tied to a specific predefined structure of the solution (Guo *et al.*, 2005).

Feature creation based on GP has been applied to problems in different areas such as pattern recognition (Pal and Wang, 1996) and data mining (Otero *et al.*, 2003), to mention a few. The purpose of *feature creation* is to generate features that represent in a concise form the problem at hand. In the pattern recognition field, the generated features are used as inputs to the classification system, in order to increase its performance. The boost in performance is obtained because the formed attributes reveal intrinsic relations between original raw features that typical classifiers are not able to find out.

Problem Statement

Methods like the combination of classifiers have been proposed to increase the recognition rate over single classifier strategies. This is the case of EoC which can obtain similar, and sometimes better, performance than a single and complex classifier. This alternative generates a trade-off between simplicity and performance (Tremblay, 2004). Several

approaches have been proposed to train EoC. A particularly interesting approach is based on the generation of subsets of features starting from an original set. This method is called *random subspaces* (RS) (Ho, 1998b). GA and genetic programming are applicable to complex and poorly defined recognition problems. Their utilization for this type of optimization problems is justified because they generate powerful EoC. The EoC obtained then consist of several individual classifiers who relay on a restricted number of features. Consequently, they have a partial representation of the recognition problem, which implies that the addition of new knowledge will allow a possible increase of the individual performance of the base classifiers of EoC, which will be reflected on an increased reliability of EoC.

Goals of the research

The main objective of this research is to study the evolutionary algorithms for the generation of relevant features applicable to different pattern recognition problems, i.e.: handwritten digits and ship recognition problem. In particular we will use genetic programming to create useful features to generate classifiers and EoC. The results are to be compared to nearest neighbours classifiers and ensemble of these classifiers generated by the RS method.

The specific objectives are:

- To present an overview of the application of GA and genetic programming for feature creation;
- To present an overview of ensemble creation methods;
- To present and develop an experimental protocol to build classifiers and EoC based on evolutionary techniques.

Organization of the document

The rest of this document is organized as follows. Chapter 2 provides a synopsis of the application of GA and genetic programming to feature selection and creation in the domains of pattern recognition and machine learning. Chapter 3 outlines the basic theory concerning the construction of ensembles, the methods for combining classifiers and the application of GA and GP to generate ensembles. In Chapter 4, the method of creating one feature at a time is presented in detail and an experimental protocol is developed to test the method. Some improvements to the algorithm are developed in chapter five: a specific mechanism to identify the over-fitting has been implemented and tested. Moreover, different alternatives to parallelize Bot's GP-based method are presented and one of them, called the *Island* method is fully implemented. Practical advantages of over-fitting control and parallelization are tested by means of an experimental protocol developed for that purpose. Once the strategy of creating features to generate classifiers is optimized, we apply it different real world problems such as a handwritten digit recognition problem using the NIST-SD19 data set with the aim of building EoC. As in previous chapters, a specific experimental protocol is also developed for this purpose in Chapter 6. Conclusions and recommendations are then presented. The thesis ends with the bibliography used throughout this research.

CHAPTER 2

FEATURE SELECTION AND CREATION WITH EVOLUTIONARY ALGORITHMS

Objects in Pattern Recognition problems are defined by a set of features useful to distinguish different categories. In complex problems, a huge number of attributes are required to differentiate between categories i.e.: classification of images of handwritten digits, which are described by 132 features in the system proposed in (Oliveira *et al.*, 2002, Oliveira, 2003b) or classification of cancer based on micro-array DNA which are described by thousand of characteristics (Hong and Cho, 2006). To manage these types of problems, researchers try to find out a feature subset that attains similar results as when the whole set is used, but reducing the amount of resources and time required. We are then looking for a subset of features that permits differentiation between categories. A complementary approach consists in creating features from a starting set, so that the resulting classifiers have a similar or better performance.

The purpose of this chapter is to explain how evolutionary techniques, in particular GA and genetic programming, can be applied to the problem of feature subset selection and feature construction. We show why genetic programming is very useful to tackle the problem of feature creation, in order to generate classifiers. The resulting classifiers (explained in this chapter) will be the constituent elements of EoC, which is the subject of Chapter 3. The current chapter is structured as follows: in Section 2.1, we introduce the concepts related to the feature selection problem. Then, a short theoretic support about GA and genetic programming is presented to facilitate the reading in Sections 2.2 and 2.3. Following Sections (2.4 and 2.5) describe different methods based on genetic methods to do feature selection and feature creation. Thus, possible alternatives to use for feature creation are established. Section 2.6 justifies the method selected to construct features for classifiers and ensembles.

2.1 Feature subset selection

Objects in pattern recognition and machine learning fields are described as set of attributes. These attributes are measurements taken of the objects. Intuitively, the more attributes used, the better the description of the object. But this statement is not always true, because there could be irrelevant, redundant or even useless attributes (e.g.: insurance numbers of patients in medical databases to classify patient as healthy or not). Furthermore, in most real-world problems, the relative importance or utility of each attribute is not known *a priori*.

Descriptions of complex phenomena use large number of features, which makes difficult subsequent steps in a classification system. An intermediate step that selects the most important features toward classification could improve not only the final performance, but also, the measurement process if the feature selection phase reveals the attributes that are ineffective for classification purposes. The problem of feature selection can be stated as follows (Devivjer and Kittler, 1982): given a set of N features,

$$X = \{x_i | i = 1, 2, ..., N\} \tag{2.1}$$

select the best subset Xs of Ns features ($Ns < N$)

$$Xs = \{x_j | j = 1, 2, ..., Ns; x_j \in X\} \tag{2.2}$$

which represents the pattern. The best subset means the combination of Ns features which optimizes a criterion function J(.), ideally the probability of correct classification, with respect to any other subsets

$$\mathrm{E} = \{\xi_i | i = 1, 2, ..., Ns\}, \tag{2.3}$$

of Ns measurements taken from X. Xs satisfies

$$J(Xs) = \max_{E} J(E) \tag{2.4}$$

Evaluation of all possible C(*N, Ns*) subsets becomes prohibitive even for small values of *N*. For *N*=12, approximately 2.7 million subsets must be evaluated (Jain et al., 2000).

Different searching algorithms for feature selection have been developed in pattern recognition and machine learning fields. Devijver and Kittler present a description of conventional algorithms: branch-and-bound, sequential forward selection (SFS), sequential backward selection (SBS), "plus 1 – take away r" in (Devijver and Kittler, 1982). Sequential forward and backward floating search methods (SFFS, SBFS) and variations to some other algorithms are analyzed and compared in (Kudo and Sklansky, 2000). Under some assumptions, Branch-and-bound method can produce an optimal subset. The other methods provide suboptimal selection techniques, which handle a trade-off between optimality and computational efficiency to make the decision (Jain et al., 2000). Jain and Zongker analysed the mentioned methods for a problem of land use classification based on satellite images and arrived to the conclusion that sequential forward floating search (SFFS) has a performance close to branch-and-bound with less processing required (Jain and Zongker, 1997, Jain et al., 2000).

Some works (Ferri *et al.,* 1993; Kudo and Sklansky, 2000) analyze the applicability of GA to feature subset selection and conclude that they are specially suited for problems of large dimensionalities, where most of the classical methods require prohibitive quantity of computations (Ferri *et al.,* 1993).

2.2 Genetic Algorithms

GA as a solution to complex systems were introduced by John Holland in 1975. They are adaptive techniques inspired by mechanisms of natural evolution. In nature, individuals compete between them for resources such as food, space and mates, such that, stronger individuals tend to survive. This is called "survival of the fittest" (Srinivas and Patnaik,

1994), and therefore "genes" of the stronger individuals tend to be disseminated. Genetic information contained in individual's genes will be inherited by his descendents. This process happens at the reproduction stage. The cycle continues with his descendents.

In GA, individuals can be represented as a bit strings (binary representation). A measure called *fitness function* is used to assign a fitness value to each individual in the population. Transmission of information from one generation to the other is done through "crossover" and "mutation" operations, inspired by their genetic equivalent in the nature. Individuals selected for these genetic operations, using their fitness values, produce new individuals called *offsprings*. New individuals will face the same process and at a certain moment, survival individuals will constitute the solution to a specific problem.

Algorithm 1 shows the structure of a Simple Genetic algorithm. For generation g=0, an initial population P(g) is randomly created and then evaluated according to a fitness function. If the termination criterion has not been reached, a new set of solutions is searched (next population) by operations of crossover and mutation. These genetic operations are applied on selected individuals P'(g), which are evaluated against a fitness function. Then, the new population for generation g+1, P(g+1) is created as a function of P(g) and P'(g). This last loop is repeated until a stop condition is found (solution desired, approximation reached or number of generation reached).

Algorithm 1: Simple Genetic Algorithm ( )

```
1  g=0;
2  Initialize population P(g=0);
3  Evaluate population P(g=0);
4  while (termination criterion not reached) do
5      g=g+1;
6      P'(g) = select solutions for next population;
7      Perform crossover and mutation operations (P'(g));
8      Evaluate population P'(g);
9      P(g) = New population(P, P'(g));
10 end-while
```

The most important details of GA are explained in the following lines. Additional information about GA can be found in the references mentioned in this section as well as in (Holland, 1992; Goldberg, 1989).

2.2.1 Population: representation and set up

Information in individuals should be coded in a way that is useful for subsequent treatment. For instance, features of individuals can be coded in a binary string. Attributes could have the same length of representation or different, but the whole representation of an individual must be a fixed number of bits. Although the binary representation is most frequently used, other representations can be more natural for specific application problems (Herrera *et al.*, 1998): vectors of integer numbers or floating point numbers for function optimization cases, ordered lists for schedule optimization problems, lisp expressions for evolving computer programs in control tasks. In addition, bounds of the range of values of each attribute have to be considered to find feasible solutions. If this is not the case, some attributes could take mathematical values that have no meaning in the real-world problems. As mentioned before, the initial population is a set of individuals (chromosomes) randomly selected. The number of individuals of the population constitutes an input parameter.

2.2.2 Fitness function

For each problem to solve there should be a fitness function to optimize. The fitness function provides a mechanism for evaluating each individual of the population and assigns a scalar value (Srinivas and Patnaik, 1994). The fitness function normalizes the objective function values, which is domain dependent, to an independent scale (for instance 0 to 1, or 0 to 100) and therefore the value that it assigns to each individual in the population (Srinivas and Patnaik, 1994). Fitness values of the individuals reflect which one is closer to the desired solution. For instance, in a maximization problem, if we like to compare two individuals, we compare their fitness values and the one with greater value is closer to the desired solution.

2.2.3 Selection methods

Selection methods are the mechanisms to replicate the nature's principle of survival of the fittest. The selection method designs how to choose the individuals in the population that will create offspring for the next generation, and how many offspring each will create (Mitchell, 1999). The selection mechanism consists of two steps: selection probability calculation and sampling algorithm (Herrera *et al.*, 1998). In the first step, for each individual in the population P, a probability of including a copy of the individual is calculated, also called "expected value" of the individual. This probability is a function based on the individual's fitness value. The second phase, sampling algorithm, produces copies of individuals to form the intermediate population P'. In this way, fittest individuals have a greater option of surviving, while weaker individuals perish (Srinivas and Patnaik, 1994). We present three different selection methods and introduce the concept of replacement strategies.

2.2.3.1 Proportional selection

In the proportional method, a survival probability p_i is assigned to each individual i $(i=1,2,...,M)$ in the population of size M. Individual i is represented by its chromosome CHi. The probability is proportional to the individual's fitness value f_i within the population:

$$p_i = \frac{f_i}{\frac{1}{M}\sum_{j=1}^{N} f_j}, \tag{2.5}$$

where $f_a = \frac{1}{M}\sum_{j=1}^{N} f_j$ is the average fitness value of the population (Srinivas and Patnaik, 1994; Hererra *et al.*, 1998).

Roulette wheel is one of the techniques used to implement proportional method. It is based on the division of a circle in sectors with angles proportional to $2\pi f_i/f_a$. Then, an individual is allocated an offspring if a randomly generated number between 0 and 2π, falls in the corresponding sector (Srinivas and Patnaik, 1994).

2.2.3.2 Tournament selection

In the simplest variant, *binary* tournament, two individuals are randomly selected from the population (with or without replacement). The one with the highest fitness value is copied into an intermediate population P' called *mating pool.* This procedure is repeated until the mating pool is full (Beasley *et al.*, 1993). Extensions of this method randomly select more than two individuals. Larger tournaments increase the selection pressure because individuals with low fitness values are less likely to win the tournament and conversely, individuals with higher fitness values are more likely to win the tournament (Beasley *et al.*, 1993).

2.2.3.3 Rank selection

Individuals are sorted in order of raw fitness and a reproductive fitness value is assigned according to the rank (Beasley *et al.*, 1993). Rank selection prevents too-quickly convergence, because the expected value of each individual depends on its rank rather than on its fitness value (Mitchell, 1999). Rank reduces the selection pressure when the fitness variance is too high ands also keeps it when the fitness variance is too low (Mitchell, 1999).

2.2.3.4 Generational and Steady-state replacement

Along with the selection of parents to build the next population, there is the concept of replacement and generation gaps: what proportion of the individuals in the current population is replaced by offspring. The concept of generation gap is related to the notions of non-overlapping and overlapping populations. In a non-overlapping population, all parents are replaced by offspring and so there is no competition between them (Sarma and De Jong, 1997). This is the conventional replacement scheme in GA and is called *generational.* In the overlapping version, parents and offspring co-exist and compete between them. This replacement scheme has been named *Steady-State* GA. Generation gap is referred as the proportion of individuals in a population that are replaced in each generation (Beasley *et al.*, 1993). A more detailed description of steady-state GA is presented in Annex I.

2.2.4 Operators

Once selection is done, the intermediate population P' is built (see Algorithm 1) and then genetic operators' crossover and mutation are applied to produce offsprings. These operators are briefly described in the following sub-sections.

2.2.4.1 Crossover

The crossover operator enables sharing of information in the population by combining genes from two parent chromosomes with the hope of producing better chromosomes (Herrera *et al.*, 1998). Pairs of individuals from the mating pool P' are randomly selected to perform the crossover operation. Each individual has a fixed length L (bits or real-valued vectors). Similarly, the crossover point within the chromosome is randomly selected. The two parents will produce two new individuals called offsprings with genes to the right of the crossover point, interchanged (Srinivas and Patnaik, 1994) as shown in Figure 2.1. If the produced string is a bad solution, it is probable that it will be eliminated in subsequent generations. In the other hand, good solutions will get higher possibilities of crossing with other good solutions in the future. Crossover operation is not done in all pairs in the mating pool. If a randomly generated number in the range 0-1 is greater than p_c (crossover rate), crossover operation is done. There exists one point and two points crossover that follows the same principles explained here. Some other variants, multipoint and uniform crossover are analyzed in (Benahmed, 2002).

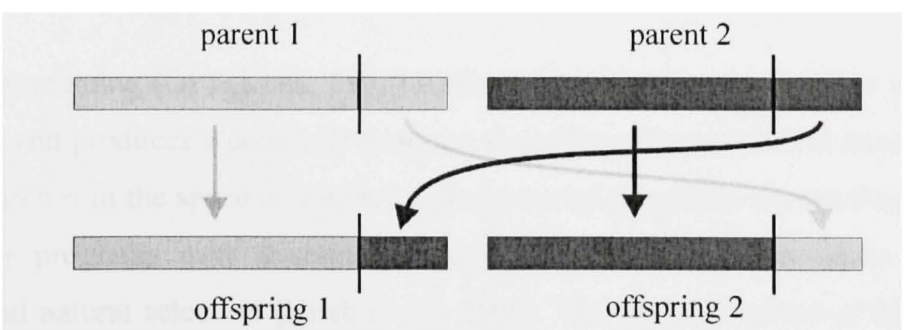


Figure 2.1 ***One point cross-over operator in GA (Kubalik, 2000).***
Information from two parents is exchanged to form two offsprings

2.2.4.2 Mutation

A common mutation operator for strings consists in inverting bits: if the number is 0, it is changed to 1 and the opposite. Mutation is done according to a probability of mutation p_m of each bit (bits of a string are independently mutated (Srinivas and Patnaik, 1994)). It ensures that the probability of reaching any point in the searching space is never zero (Beasley *et al.*, 1993). Mutation keeps diversity in the population (Tay *et al.*, 1997), because options in the neighbourhood of good solutions are explored (alteration of one bit in the chromosome). This means that mutation is able to restore lost or unexplored genetic material into the population and it prevents the premature convergence of GA to suboptimal solutions (Herrera *et al.*, 1998). Figure 2.2 illustrates the mutation principle.

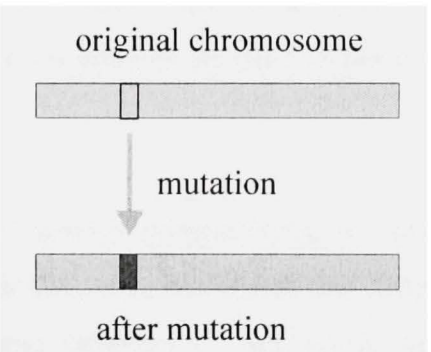


Figure 2.2 ***Mutation operator in bit string GA. (Kubalik, 2000).***
One gene (bit) is inverted. Mutation helps to generate novelty in the population

2.3 Genetic programming

Genetic Programming (GP) (Koza, 1992) works with a high level statement of a problem's requirement and produces a computer program that solves the problem at hand (Hirsh *et al.*, 2000). It searches in the space of computer programs by progressively breeding a population of computer programs over a series of generations following Darwinian principles of evolution and natural selection (Hirsh *et al.*, 2000). The general outline of GP is similar to GA: the initial run of GP proposes a group of randomly created computer programs. Then, it evaluates the created programs to determine which is better to solve the problem. GP selects

probabilistically programs from the population based on a ranking established and modifies these programs by executing cross-over and mutation operations (Hirsh *et al.,* 2000). This cycle is repeated until a stop criterion is met. Typical stop criteria are: optimal solution is found or the number of generations is reached. John Koza introduced tree-based genetic programming in his book "Genetic Programming – On the programming of computers by means of natural selection". Figure 2.3 shows a flowchart adapted from (Koza, 2004; Koza, 1992). For generation zero an initial population is randomly created. If this population does not satisfy the termination criterion, the fitness measure is applied to each of the *Pop_Size* individuals in the population. Afterwards, the selection operator is applied to choose the individuals that take part in reproduction, cross-over and mutation operations according to the corresponding probabilities. This is done to produce the population for the next generation. The new population follows the same steps of fitness measure and genetic operations until the termination criterion is satisfied (a pre-defined number of generations or a solution performance).

The most important details of genetic programming are presented in the following lines. Because of its similarities with GA, only the points that differ are explained in some detail. Initially we explain the general structure of individuals and then some types of fitness functions currently used in the genetic programming literature. Selection methods are based on fitness values, so the description from Sections 2.2.3 is also applicable to genetic programming. Genetic operations entail some differences against GA, and therefore they are explained here. Additional information about genetic programming can be found in (Koza, 1992; Banzhaf *et al.,* 1998; Langdon, 2000).

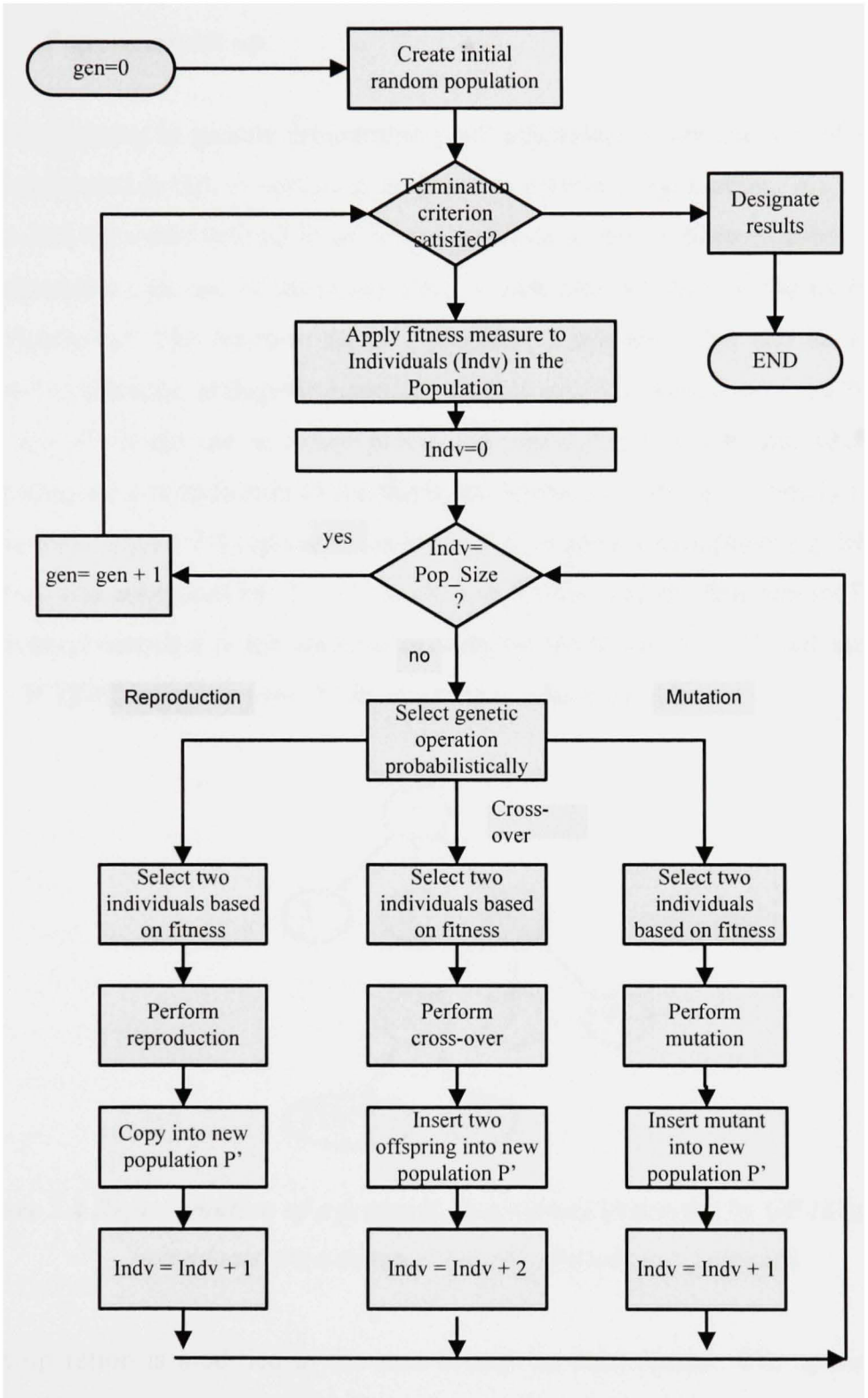


Figure 2.3 ***Genetic programming flowchart. (Adapted from Koza,1992).***

2.3.1 Population set up

Computer programs in genetic programming are equivalent to the concept of individuals or chromosomes used in GA. A computer program, in genetic programming, is a composition of functions and terminals defined in advance according to the problem to solve. The terminal set corresponds to the set of attributes used to describe the data of the problem, so it is domain dependent. The function set is composed of operators that can be applied to the terminals. For instance, arithmetic operators such as addition, subtraction, multiplication and division are of current use in image processing applications. Since, individuals in genetic programming are a composition of functions and terminals; they are normally represented in a tree structure. Figure 2.4 represents an individual in genetic programming. In this case the terminal set T is composed by: T = {1, 2, 10, 3, 4, TIME} and the function set F= {+, IF, >}. The individual outcome is the addition of terminal elements "1", "2" and the result of IF operator. If TIME is greater than 10, the result is 6, otherwise is 7.

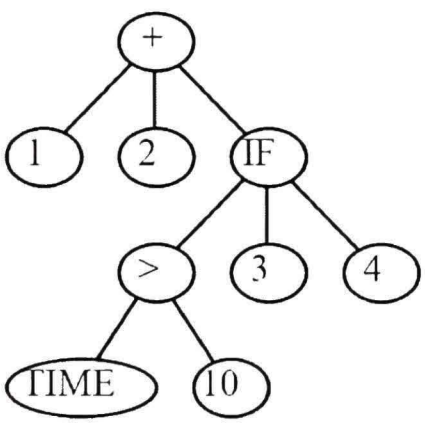


Figure 2.4 ***Representation of a program (individual) generated by GP (Koza, 1992).***
Individuals are a composition of terminals and functions

Division operation is modified in the case of zero as denominator. The operation is called "protected division" and is defined as a constant (could be zero or one) when the denominator is zero. For other values of denominator, protected division works as the standard arithmetic operation. Standard logical operators are also used: AND, OR, NOT,

XOR. Construction of the trees must follow certain rules to produce valid outcomes. In the case of Figure 2.4, the IF operator requires the element on the left to be a comparative operator and the two terminal elements on the right are numbers or variables. This is referred as "closure property" of the function and terminal sets (Koza, 1992).

2.3.2 Fitness function

The objective function to be optimized is defined according to each problem. It is the driving force of the evolution (Koza, 1992). Hence, the fitness function assigns a scalar value to each individual of the population, which reflects how good the individual is to solve the problem. In the same way, fitness values control the application of operations that modify the structures in the population (Koza, 1992). In general, fitness measure must be representative of the domain space as a whole, because it forms the basis for generalizing the results obtained to the entire domain space (Koza, 1992).

An example of fitness measure used throughout our experimentations is the recognition rate. The individual (i.e.: tree) is interpreted by using the terminal and operators sets and thus serves as input to a classifier. The recognition rate of the classifier corresponds to the fitness measure. There are different measures used to quantify the individual performance: raw fitness, standardized, adjusted and normalized (Koza, 1992). Recognition rate corresponds to a raw fitness. Standardized fitness restates the fitness measure so that lower numerical values mean better fitness values; for instance, error rate instead of recognition rate. Adjusted fitness creates a function that maps the range of values to the interval 0 to 1, and bigger values are better. In normalized fitness, the sum of population fitness equals 1, on top of the characteristics of a standardized fitness. Additional details can be found in (Koza, 1992).

2.3.3 Operations in Genetic Programming

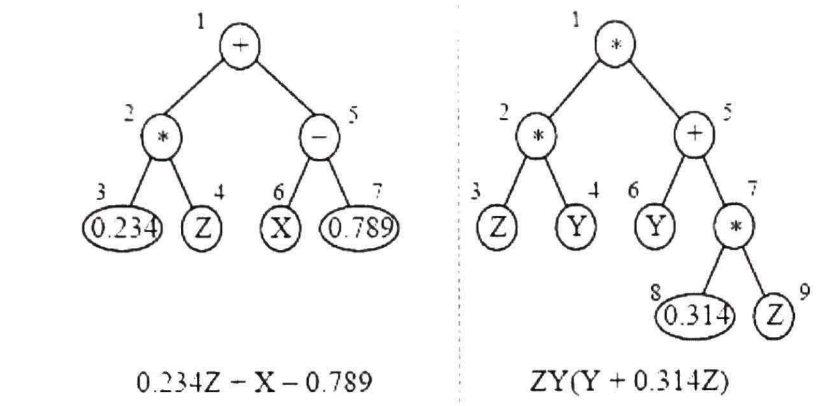
Mutation and crossover (sexual recombination) are the operations derived from nature. In the field of GA, the application of these operations is straightforward because GA has a well defined structure that is conserved during these operations. In contrast, in genetic programming, the structure could be changed with these operations, as is presented above.

2.3.3.1 Initialization operator

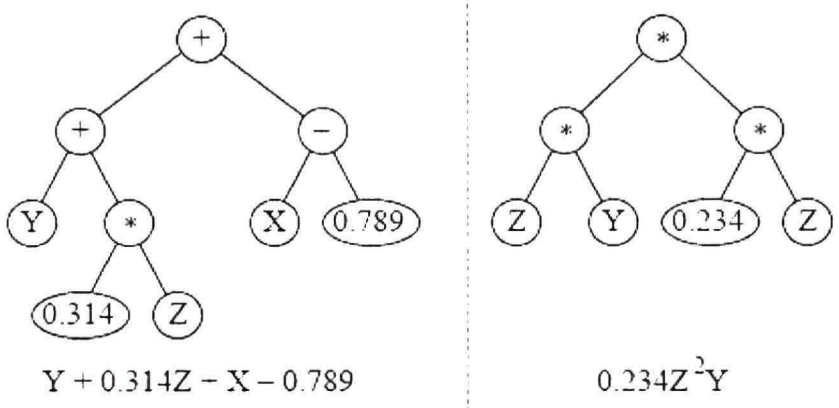
This operator determines how the initial population is produced. The aim is to create the best random trees (individuals) for the problem at hand. The initialization operators depend on allowed tree size parameters. These parameters have to be carefully set to avoid premature convergence and the apparition of *bloat* (individuals grow uncontrollably until the maxima allowed depth and size). The most common methods are full, grow, ramped-half and half (Koza, 1992). The "full method" creates trees with each path with a length equal to the specified depth (Koza, 1992) (path goes from root to the end-point). The "grow method" creates trees of various shapes with lengths of each path no greater than the specified length (Koza, 1992). Ramped-half and half creates an equal number of trees of different depths and, for each depth value, 50% of the trees are created via the "full method" and 50% via the "grow method" (Koza, 1992).

2.3.3.2 Crossover operator

In this operation two parents are selected based on fitness and points of crossover are randomly and independently selected in each parent. For instance in Figure 2.5 (from (Koza, 1992)), point 2 is selected from "left parent" and point 5 in "right parent". In the crossover operation, selected sub-trees are interchanged at independently selected points. The cross-over operation presented here is the version with two offspring. There are some other versions. Resulting offspring are valid executable programs.



(a)



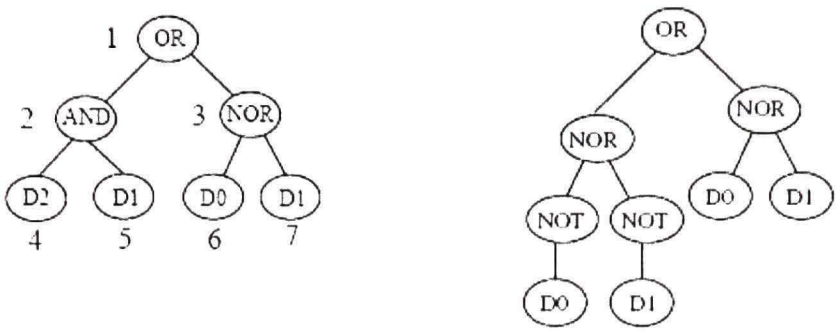
(b)

Figure 2.5 ***Crossover operation in GP with two offspring.***

Sub-trees randomly and independently selected at points 2 ("left parent") and 5 ("right parent") in (a) are interchanged to produce two offspring in (b.)(Taken from (Koza, 1992))

2.3.3.3 Mutation operator

The steps to perform a mutation operation are (see Figure 2.6, from (Koza, 1992)): a parent is selected probabilistically based on its fitness measure, then a point within its tree is randomly selected and the sub-tree at the selected point is deleted. A new sub-tree is grown at the mutation point in the same way as done with trees for the initial random population. The resultant tree is the offspring produced by mutation. It is a new program syntactically executable (Koza, 2004).



(a) Parent (b) Offspring

Figure 2.6 ***Mutation operator.***

Point 2 is picked within the selected parent for mutation operation (a). Sub-tree deleted and a new sub-tree is grown at this point. Resultant offspring is in (b). Taken from (Koza, 1992)

2.4 Feature selection and construction with genetic algorithms

GA have proved to be successful in optimization problems where the searching space is large and poorly defined. Some authors have analyzed the application of GA to the subset selection search problem. Ferri *et al.* made an initial study about its applicability and concluded that they are specially suited for problems with medium size dimensionalities (20 to 49 according to (Kudo and Sklansky, 2000)), where most of the classical methods require prohibitive quantity of computations (Ferri *et al.,* 1993). In other study, Ferri *et al.,* compare sequential forward and GA methods. They found that GAs is a positive alternative because of its near-optimal search of the space due to its inherent randomisation mechanism, but it has a limited applicability to very high dimensional spaces (hundreds of features) (Ferri et al., 1994). Their results show that GA get trapped in good solutions (not optimal) because of its premature convergence and no further improvement is possible (Ferri *et al.,* 1994). Jain and Zongker compares GA with sequential forward floating search (SFFS) based on satellite images to land use classification (Jain and Zongker, 1997). They found difficult to establish a fair comparison because GA does not attempt to find the best subset of a specified number of features, instead its search space encompasses all subsets sizes (Jain and Zongker, 1997). In a posterior analysis of large-scale feature subset search (50 or more features), Kudo and

Sklansky found contrasting results: GA is indeed very well suited for large-scale feature problems (Kudo and Sklansky, 2000). Possible explanations for these differences, as mentioned by Kudo and Sklansky, are the set of parameters used in GA runs and the objective function that drives the GA search: for instance, the search time (or number of generations in GA language), the population size and genetic operation probabilities. They mentioned that certain amount of experimentation is required to find out a suitable range of GA's parameters. Once the right range has been set, GA's results, are superior when compared to sequential search methods and the variability of their results is very small (Kudo and Sklansky, 2000). Therefore, GA can be applied to select subset of features that improves classification performance with a reduced number of features. That simplifies the resulting classifiers, so they can be combined in ensembles. We present in the following lines an interesting approach to do feature selection and construction that restructures the initial feature space and generates classifiers with similar or better performance.

Vafaie applies GA to restructure feature-based representation spaces to different problems (Vafaie, 1997). Accuracy on a classifier system depends on the input quality and classifier quality. Input quality indicates how well the problem is represented by the initial set of features. Classifier quality indicates the intrinsic performance of the classifier independently of its input data. These two aspects take place in the final performance of the system. Vafaie proposed a method to restructure the representation space of the input data, before giving them to the classifier, so that the overall system has a similar or better performance when compared to a system using the original representation (Vafaie, 1997; Vafaie and De Jong, 1995). The general architecture applied is presented in Figure 2.7. An initial set of features is used as input to the system. The system selects a candidate feature set by means of the application of any of the GA modules: selection or construction. Determination of the module to be used is done by the module selector mentioned above. The output of the GA module is evaluated by means of an evaluation function which qualifies how good the candidate set is. This process continues with the same module until the GA converges or the predefined number of generations is reached.

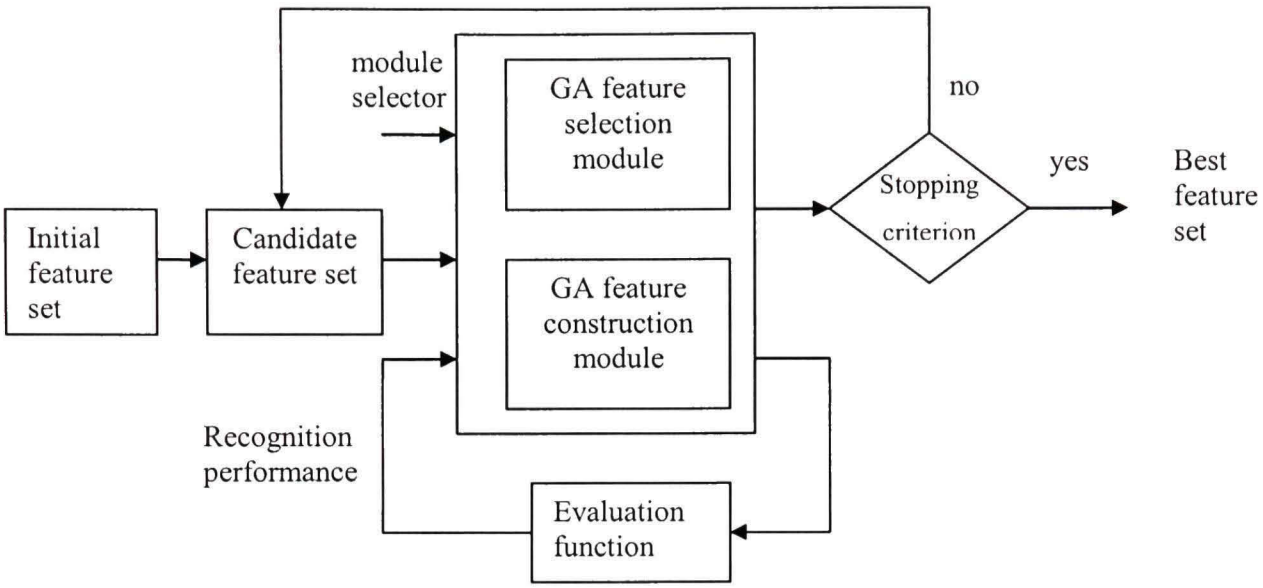


Figure 2.7 ***Architecture of combined feature selection and construction based on GA.*** *(Vafaie, 1997)*

Convergence can be associated to the system performance. Later, the other GA module is applied to the previous result obtained until a stopping criterion is reached. Vafaie uses first the GA feature selection module to reduce the number of features. Then, she applies the GA feature construction module to the result from feature the selection module. The objective of this module lies in the exploitation of interrelationships among the selected features to improve the overall system performance (Vafaie, 1997).

Transformation of the feature set continues until a repetition of a complete cycle feature selection-construction does not improve performance (Vafaie, 1997) or there is no change in the representation. The last feature set is called the best feature subset. Final performance of the best subset is measured by submitting unseen data to the evaluation function or classifier used. The system has been tested with problems with different degrees of complexity. Applicability to different problems is shown and scalability is proved with small texture sets and large eye localization data sets (Vafaie, 1997). In addition, independence of the evaluation function is given by comparing results when using AQ-14, C4.5 and nearest

neighbour (k-NN) algorithms (Vafaie, 1997). Unfortunately, several passes through the system are required to generate a feature subset that manages the trade-off between reduction of the number of features and improving or maintaining of the performance. Feature construction is applied each time after the feature selection module. As a result, the time and the amount of resources required to generate an improvement becomes huge when the complexity of the problem increases.

The way in which the GA-based feature construction module is applied, is very similar to the GP principles. Recently, various methods of feature construction based on GP have been published with encouraging results in recognition rate (performance) and number of features required. The following section presents some of these methods.

2.5 Feature Construction with Genetic Programming

GP has been applied to feature selection problems only in few cases (Muni *et al.,* 2006). Feature construction is inherent to GP: it searches functions that operate on attributes of the terminal set to solve the problem at hand. The functions used belong to the predefined function set. As the evolution progresses, GP discards the attributes that are not useful to solve the problem. Thus, by applying GP we are doing function construction and a sort of attribute selection. GA-based feature selection establishes the best combination of features toward a specific goal (for instance, classification). On the other hand, GP searches for the best combinations and attributes toward classification. It discovers functional relationships between attributes (Raymer et al., 1996), which are not easy to find with others optimization methods. A simple example, presented in (Bot, 2001), is the body mass index (BMI) that is used by insurance companies to establish the risk of heart attack of clients. The BMI index is calculated as the ratio of weight in kg to height in squared meters (British Heart Foundation http://www.bhf.org.uk/questions/index.asp?secondlevel=1164&thirdlevel=1337). In this example, selection of features height and weight is not enough to measure the risk of heart attack; it is necessary to discover the ratio weight (kg)/ height^2 (m^2). Construction of functions can be done by using GP because it is able to establish the structure and values of

the solution while GA is tied to a specific predefined structure of the solution (Guo *et al.*, 2005). Hence, GP can be a better option than GA to create classifiers based on an initial feature space, because it is able to find out the intrinsic relations between the initial feature set (structure of the solution) and the value of the solution (interpretation of the resulting tree). We present some of the methods to construct features based on GP and select the most appropriate to creation of classifiers for ensembles.

Sherrah has developed an automatic non-parametric method for the extraction of non-linear features using GP. This method, called *Evolutionary pre-processor,* automatically selects the number of features to extract, determines which of the original attributes are useful for classification, specifies the non-linear transformation useful for the problem and chooses the classification algorithm to use among a given set (Sherrah, 1997). Tests have been carried out with nine real-world data sets taken from the Machine Learning database repository of the University of California at Irvine, known as UCI database (Newman *et al.*, 1998). As a result of the application of the evolutionary pre-processor, misclassification and dimensionality were reduced. In addition, the Evolutionary Pre-processor determined automatically whether to perform feature generation or feature selection (Sherrah, 1997). The main disadvantage of the evolutionary pre-processor is its computational complexity (Sherrah, 1997), so the construction of classifiers becomes a long and complex task.

Another approach, presented in (Bot, 2001) constructs (generates) new features iteratively. The first evolved feature is constructed with a standard GP algorithm. Evolution for new constructed features considers the new feature in conjunction with the already evolved features. The new feature is included in the solution if the performance increment generated is greater than a threshold. To explain the procedure, suppose that there are *n-1* evolved features and the system is evolving the *n-th* feature. Fitness score is calculated taking into account the already generated *n-1* features and the new one (*n-th feature*). If the fitness increment is smaller that a threshold value *d,* the new generated feature (*n-th)* is taken away and the process stops. If fitness increment is higher than *d*, the new generated feature (*n-th*) contributes to the objective and is definitely included in the generated set (that now has *n*

features). The process continues until the increase in fitness is less than constant *d*. The method is tested by classifying 16 datasets from UCI Machine Learning Repository (Newman *et al.*, 1998). MDM and *k*-NN classifiers are used. Results show that representation of these datasets can be done with as low as 2 or 3 generated features and accuracy is equal or better than that obtained with the *k*-NN alone (Bot, 2001). The best characteristic of this method is its ability of representing a classifier with just a few features while maintaining its performance. This characteristic turns this method very useful to create classifiers for ensembles.

Krawiec introduces a variation to the GP algorithm towards preserving useful parts within constructed features during GP runs. Preserving means that each individual's genotype is split in two parts. Both sets participate in the evaluation process for new features but the hidden part is not considered during the evolution process (Kraweic, 2002). Individuals decide on their own, which part is hidden and which is visible for the evolutionary process. This variation to classical GP algorithm protects important features for classification purposes. Tests were done over several data sets including real and synthetic data. Results show that accuracy of the classification in tests sets is improved. In some cases has significant improvement from a statistical point of view (Krawiec, 2002). Again, the application of this method makes the process of creating classifiers long and very burdensome.

A different method to generate features is presented in (Guo *et al.*, 2005). The authors state that feature selection methods have a limited power, because searching is done over a pre-defined space. They used GP instead of GA, because they are able to establish the structure and values of the solution. In contrast, GA is tied to a specific structure of the solution. They provide raw vibration data recorded from a rotating machine with six different conditions to a GP algorithm. Created features are used as inputs to a neural network to classify the six conditions. GP generates mathematical expressions guided by a fitness function that will maximize the inter-class scatter over the intra-class scatter using the Fischer criterion (Duda *et al.*, 2001). This criterion is iteratively applied to pairs of classes. This change in the fitness

function presents an enormous advantage of reducing the calculation required (Guo *et al.*, 2005) in comparison to a fitness based on *k*-NN classifieirs. Results obtained show the ability of GP to discover automatically features that make evident the difference between bearing conditions (Guo *et al.*, 2005). These features are expressed in the form of non-linear mathematical expressions. This method is promising and employs a different classifier strategy. Unfortunately, it has been only applied to a specific application which does not guarantee its benefits to other applications. The application has a small number of classes and the resulting outcomes for each class (when interpreting the solution trees) were easily distinguished maybe because the problem itself has a small degree of overlap between classes.

2.6 Selection of the feature construction method

GP and GA are searching methods applicable to feature construction and particularly advantageous for this type of problems. In this section we justify the selection of the feature construction method to build classifiers and ensembles.

Unlike other searching techniques such as hill climbing and gradient search, genetic search algorithms are not greedy (Koza *et al.*, 2005). In addition, the recombination operation (cross-over) works in conjunction with the population in a way that makes it a good approach for feature creation.

Most of the searching and optimization techniques used in machine learning and artificial intelligence fields are based on greedy methods that move from a position to another position of the searching space (the new point is only accepted if it is superior). GA and GP sometimes accept points with inferior performance than current points (Koza *et al.*, 2005). We refer here to points because genetic mechanisms use a population of solutions, that is, there is a parallel searching. Inferior points are accepted because exploration is based on a probabilistic Darwinian selection process (Koza *et al.*, 2005). In such way, best individuals

are not guaranteed to be selected and worst individuals are not necessarily excluded (Koza *et al.,* 2005) from the searching process.

In addition to parallel search not being exclusively based on best individuals, genetic methods use a mechanism to transfer the information between the different threads so that the general view of the searching space promotes the location of the global optimum (Koza *et al.,* 2005). Furthermore, genetic operations improve the options to find the global optimum. Mutation causes a small change in the individuals that enables restoration of lost genetic material or extends the search to closer points that can further improve the current solution. Cross-over is applied to performing individuals (they have certain positive information toward the solution of the problem that helps them to be selected for the operation). This interchange of information can yield a better solution. In conclusion, genetic operations promote the interchange of useful information, which helps finding the global optimum. This makes genetic methods very promising for feature construction.

GP has an additional advantage over GA because they are not tied neither restricted to the pre-defined structure of the solution. GA search for a solution in the search space defined by the structure of the individual's chromosome. GP is able to construct itself the structure and the values of the solution. Both mutation and cross-over operations can alter dramatically the structure of the solution towards more promising regions of the searching space. All these reasons support our selection of GP as the tool to use in building classifiers.

We presented different methods to create features for classifiers in Sections 2.4 and 2.5. In this section we support the idea of using GP to create these classifiers. Between the methods presented in Section 2.5, the approach that generates more flexible and performing solutions is the Bot's method. Flexible because we can choose the number of features of the solution in terms of the performance desired.

CHAPTER 3

ENSEMBLE OF CLASSIFIERS

3.1 Introduction

Classification within pattern recognition aims to build classifiers with high recognition rates. Therefore, classifiers are refined by means of different strategies. A natural approach consists in improving the performance of the classifier. Increment of the classification performance is carried out at expenses of complexity and degradation to certain sample types or even classes. It has been observed (Kittler et al., 1998) that patterns misclassified by different classifiers do not necessarily overlap (for different classifiers, not always the same samples are misclassified). What happen if these classifiers are used at the same time to classify the data at hand? Can the combination of classifiers yield better results than the single best classifier? This scheme of multiple classifiers increases the complexity of the system, but it has been proved that performance can be greatly incremented if adequate classifiers and parameters are chosen for the combination (Ho, 1998a; Kuncheva, 2004). To design EoC with higher accuracy, individual classifiers have to be accurate and diverse (Hansen and Salomon, 1990). Dietterich mentions that three fundamental reasons explain why it is often possible to construct very good ensembles: *statistical*, *computational* and *representational* (Dietterich, 2000). The statistical reason bases its benefits on the idea that an operator applied to the combination of classifier's responses minimizes the risk of picking an inaccuracy single classifier. If a single classifier is selected and it is inaccurate, then performance is bad. If the single classifier picked is accurate, a good response is obtained. The response of the combination of classifiers tends to be the "average" answer, then the risk of bad responses is minimized and combination could generate responses with similar good responses to the best classifiers of the ensemble (Kuncheva, 2004). The *computational* reason relies on the idea that many learning algorithms perform a search on a reduce part of the searching space and they are more susceptible to get trapped in local optima than an EoC running also in reduced parts of the space but staring from many different points in the whole searching space (Kuncheva, 2004). Finally, weighted combinations of classifiers can expand the space of

representation giving better results than a single classifier with limited representation capabilities. This *representational* reason is based on the idea that normally there is no single classifier that can represent the true hypothesis while a combination of classifiers such as a weighted sum of classifiers' hypothesis expand the space so that the combination may be a more accurate approximation to the desired hypothesis (Dietterich, 2000).

3.2 Design of ensemble of classifiers

The main elements that take part in the design of EoC are: data, classifiers and the combination functions. Data samples can be split between different classifiers or the attributes (features) within data can be divided in different sub-sets. Classifiers used in the ensemble can be of the same type or different types. Different combination functions can be used to generate the ensemble response. According to the mode in which data samples, classifiers types and combination functions are used, different methods for constructing ensembles are derived. We summarize the following methods for constructing ensemble of classifiers: methods that manipulate the training examples, methods that manipulate the input features, methods that manipulate the output targets and methods that manipulate the members of the ensemble (Dietterich, 2000; Dos Santos, 2004).

3.2.1 Constructing ensembles by manipulating the training examples

The manipulation of training samples aims to generate multiple hypotheses (Dietterich, 2000). The same type of classifier is run several times, each time with a different subset of training examples and then their answers are combined. Dietterich indicates that this technique works well on unstable learning algorithms such as decision-trees, neural networks and rule learning algorithms (Dietterich, 2000). Some other techniques as linear regression, nearest neighbour are very stable against manipulation of the training samples (Dietterich, 2000). Within this group, there are two classical approaches: Bagging and Boosting. In bagging techniques, the classifiers are trained independently with T_B different replications of the training set. N_B samples are randomly drawn with replacement, from the original set of

Nbr_Samples samples (Dietterich, 2000), thus, some samples may appear more than once and some other may not appear at all. The ensemble is the aggregation of the T_B classifiers and all of them participate to classify new data. The class predicted most often by the T_B classifiers is assigned to the sample tested, with ties solved arbitrarily. Breiman states in "Bagging predictors" (Breiman, 1994) that bagging will improve accuracy with learning algorithms that are unstable. He refers as unstable algorithms, those in which small changes in the data produce large changes in classifier performance (Zhang and Bhattacharyya, 2004).

Adaboost is another classical representative algorithm of boosting method (Dietterich, 2000). It generates multiple hypotheses manipulating the training samples (a set of weights are maintained over the training set). In each iteration the learning algorithm attempts to minimize the weighted error on the training set and returns a hypothesis. Change of weights aims to place importance on samples that were misclassified in previous iterations and less weights on samples correctly classified. The final classifier is constructed by a weighted vote of the individual classifiers, and other classifiers are weighted according to its accuracy on the weighted training set (Dietterich, 2000).

3.2.2 Constructing ensembles by manipulating the input features

This method provides a partial view of the data to each classifier of the ensemble. Partial view because each classifier has only a subset of the features available in the training data. As each classifier has a reduced set of features, the effects of the *curse of dimensionality* problem are also diminished (Ho, 1998a). Random Subspace (RS) method is a traditional representative of this group. This method relies on a stochastic process that randomly selects a number of features of the given set to construct each classifier (Ho, 1998a). The stochastic method used, introduces independence, to a certain extent, between classifiers (Ho, 1998a). Ho assures that constructing systematically the classifiers as indicated and combining them, can achieve very high accuracy.

The method can be formulated as follows (Ho, 1998a): given a set of *Nbr_Samples* examples in an *N*-dimensional space

$$\{(x_1, x_2, x_3, ..., x_N) \mid x_i \in \Re, 1 \le i \le N\}, \tag{3.1}$$

N_{RS} -dimensional subspaces are considered

$$\{(x_1, x_2, x_3, ..., x_N) \mid x_j = 1, j \in I, x_j = 0, j \notin I\}, \tag{3.2}$$

where I is an N_{RS}-element subset of $\{1,2,3...,N\}$, and $N_{RS} < N$. For each constructed classifier, a subspace is chosen by randomly selecting an I from $C(N, N_{RS})$-many choices. All *Nbr_Samples* examples are projected onto the chosen subspace.

The method, as proposed by Ho, is a derivation of stochastic discrimination where many stochastically weak classifiers are combined to get nearly monotonic increase in accuracy (Ho, 1998a; Ho, 1998b). Ho declares that individuals classifiers constructed in this way do not have full discriminative power but the combination of these weak classifiers generalize very well to unseen data of the same problem. The high accuracy reached depends on the statistical properties of the combination function used, but close values to the high accuracy are obtained well before a large number of weak classifiers are combined (Ho, 1998b). The stochastic procedure of RS introduces independence among the classifiers, thus combining their decisions gives the ensemble its discriminative power (Ho, 1998a; Ho, 1998b).

Ho shows the utility of RS method applying to different classifiers and problems. In (Ho, 1998b), the RS method is applied for constructing decision forests (multiple decision trees) and it is tested on different datasets from UCI repository (Newman *et al.*, 1998). She compares the results against single-tree classifiers and other forest construction methods. Superiority of the method is clear when the dataset has a large number of features and examples (Ho, 1998b). Accuracy on training data is maintained and generalization power increases as complexity grows due to combination of classifiers (Ho, 1998b). Ho also built

ensemble of *k*-NN classifiers based on RS method and tested it with a handwritten digit dataset from U.S. Postal Services (Ho, 1998a). She showed again that accuracy is improved nearly monotonically with the addition of new component classifiers. The ensembles outperforms of *k*-NN classifier using the entire feature set and prototypes. Superiority of the ensemble is maintained even when decreasing the number of training samples (Ho, 1998a). RS method generates good results for problems with a relative large number of initial features. In the case of a small number of features, Ho proposes to increase the number by using simple functions like pair-wise sums, differences and so on (Ho, 1998a).

EoC constructed by the RS method have a double advantage: they only use a small number of features from the original set and even though, they improve the performance in comparison to an ensemble that uses the whole feature set.

Tremblay *et al.*, built ensembles of *k*-NN classifiers constructed with RS method and then used performance and diversity measures to optimize the creation of ensembles of classifiers. GA (single and multi-criterion) were used to search the best ensemble of *k*-NN (Tremblay *et al.*, 2004).

3.2.3 Constructing ensembles by manipulating output targets

Previous methods presented manipulate the input patterns set in some way. A different approach consists in managing the original classes in modified sets. The method is presented and reviewed in (Dietterich, 2000; Dietterich, 2002). This approach consists in managing the original output classes in modified sets and then training one classifier. This process is repeated L times. Classifiers are to be built to solve each new problem and, when used as an ensemble, the original problem is consequently resolved. Following lines describe the method (Dietterich, 2000): Suppose that the problem at hand has a large number of output classes *K*. Samples in the training data are randomly split in two sets A_l and B_l and relabelled to 0 and 1, and a classifier C_l is generated for the new labels. The original class labels in sets A_l and B_l are conserved. This process is repeated *L* times, so that *L* new

classifiers are trained. The L created classifiers work as an ensemble to classify unseen data. Each classifier gives its answer about the sample to classify. If the answer from classifier C_l, is 0, then every class in sets A_l receives a vote. Otherwise (answer from classifier C_l, is 1), every class in sets B_l receives a vote. Unseen samples are presented to all L classifiers and the class that receives the highest amount of votes is chosen as the class assigned to the unseen sample by the ensemble.

3.2.4 Constructing ensembles by manipulating the ensemble members

Manipulation of ensemble members refers to use classifiers of different types instead of the same type for the whole ensemble or the same type of classifiers but with different parameters (Dietterich, 2000). Opitz and Maclin mention that a simple ensemble of neural networks that differ only in their random initial weights has a good performance comparable to results obtained by using Bagging method (Opitz and Maclin, 1999).

3.3 Methods for combining classifiers

In the previous section, we have mentioned different methods for constructing ensembles. Performance of ensembles is superior to single classifiers, but we have not indicated, in all cases, the functions to use when combining the component classifiers. Combination functions also depend on the topological interconnection between classifiers. Some of the common functions are: majority voting, sum, product and weighted average (Kittler *et al.,* 1998). They are applied to a parallel interconnection between classifiers. The first consists in taking as output class for the tested sample, the class that receives the majority of votes from the classifiers of the ensemble (Dietterich, 2000; Ho, 1998a). This is applicable in the case where only the output class labels are available (Kittler *et al.,* 1998). In other cases, when real values like posterior probabilities are available, linear combinations can be used: sum, average, weighted average (Kittler *et al.,* 1998). Some other combination functions like product, maximum and minimum can also be used (Kittler *et al.,* 1998). Lam presents and analysis of different interconnection topologies and its effects on the ensemble (Lam, 2000).

Once topology connection and combination functions are defined, the ensemble generated is ready to be applied. Some questions relate to the number of classifier to use arise. Is it better to utilize all the available classifiers? Does the performance improve monotonically as the number of component classifiers grows? Opitz and Maclin tested the impact of the number of classifiers in the ensemble has over the performance for different types of ensembles. They worked with 23 datasets from UCI repository (Newman *et al.,* 1998) using neural networks and decision trees as classification algorithms (Opitz and Maclin, 1999). They conformed ensembles up to 100 classifiers and found that for bagging and boosting methods applied to neural networks, much of the reduction in error is obtained with the first ten to fifteen classifiers (Opitz and Maclin, 1999). When the classifiers used were decision trees, there were needed 25 classifiers to attain a plateau. Some other questions come up: From the pool of classifiers (i.e: the 100 in the case of Opitz and Maclin), which ones have to be used to minimize the composed error of the ensemble? Do we take the 15 or 25 best classifiers? To answer these questions we arrive to the problem of classifier selection for building ensembles. Following section describes with this topic.

3.4 Ensemble of classifiers and evolutionary algorithms

In preceding sections some of the classical methods for constructing ensembles of classifiers have been described. A general conclusion was that the performance of the ensemble is better than the performance of the single best base classifier. The performance of the ensemble can be improved further in different ways: (i) selecting some classifiers of the ensemble and building a new ensemble to obtain similar or better results; (ii) improving the performance of the base classifiers and obtaining, as a result, a new ensemble with better performance. In both cases, the purpose is to improve the performance of the whole ensemble, but the approach is different. Following subsections describe these approaches and discuss the interrelation between EoC and evolutionary algorithms. In the first case, the classifiers are already built and we search for the best combination of base classifiers, while in the second case, we like to build base classifiers with high performance so that the resulting ensemble yield good results.

3.4.1 Selection of classifiers and ensembles

Tremblay did an exhaustive study of selection of *k*-NN classifiers to optimize ensembles generated by the RS method (Tremblay, 2004; Tremblay *et al.,* 2004). The target was to select the ensemble of *k*-NN classifiers with the highest performance and the lowest cardinality (the ensemble with fewest classifiers is considered the simplest). Two aspects have to be considered to select the best set of classifiers: the searching algorithm and the selection criteria. An initial search space is composed by 100 *k*-NN classifiers. Such a huge searching space requires a searching algorithm that can explore large and poorly understood spaces and GA have this ability. Having in mind that diversity is one of the conditions to build successful ensembles, the initial selection criteria used was to jointly maximize performance and diversity of the ensemble and therefore multi-objective GA are used. Ensemble's recognition rate of test data sets was used to measure the ensemble's performance and various typical diversity measures were studied. Tremblay arrived to the conclusion that none of the diversity measures used provides a real advantage in searching performing ensembles. On the other hand, he found that single GA provides ensembles with the highest performance without much scarifying the cardinality of the ensemble (Tremblay, 2004).

Different versions of GA have been applied to search the best classifiers according to varied selection criteria. Radtke uses simple genetic and MOMA algorithms to optimize a projection distance (PD) and a multiple layer perceptron neural network (MLP) classifier (Radtke *et al,* 2006). Selection of ensemble of classifiers was done in (Dos Santos *et al.,* 2006) considering not only the ensemble's performance and complexity (cardinality) but also the over-fitting phenomenon. Again, the searching space is vast, so single and multi-objective GA are used. When using single GA, they found that performance was the best searching criterion to increase ensemble's performance (Dos Santos *et al.,* 2006). Twelve different diversity measures were jointly used with ensemble's performance to find out the best searching criteria. It was found, as in other publications, that diversity alone as searching criterion does not produce better ensembles (Dos Santos *et al.,* 20). In the other hand, when

diversity was jointly used with performance as the searching criterion (with multi-objective GA), performance of the ensemble was improved. Anyway this performance was smaller than the resultant ensemble when using error rate as searching criterion in single GA. In all cases, over-fitting was controlled by creating a strategy called global validation, in which the best solutions in a validation set are stored in an auxiliary archive generation by generation during the optimization process (evolution of the GA). This assures to find out the best possible performance in generalization without over-fitting the optimization dataset.

Diversity between classifiers is essential for successful ensembles. Then, the combination of miscellaneous diversity measures and performance has been studied (Dos Santos *et al.*, 2006; Ko *et al.*, 2006).

3.4.2 Feature selection for ensemble generation

Different approaches have been reported in the literature regarding the creation of optimized base classifiers so that performance of the ensemble is improved. Guerra-Salcedo and Whitley present in (Guerra-Salcedo and Whitley, 1999a; Guerra-Salcedo and Whitley, 1999b) a strategy to select feature subsets in each base classifier, based on GA, so that the final ensemble generates good results. They compare four different methods of ensemble creation. Two of them use the complete set of features (bagging and boosting) and two other methods that use only some of the features from the original set (RS and their method). For comparison purposes, the number of selected features is the same as the number of randomly chosen features. In their procedure, the features to include in each base classifier, are selected with a GA (Guerra-Salcedo and Whitley, 1999a; Guerra-Salcedo and Whitley, 1999b). The ensemble built that way outperforms ensembles created based only in RS method, when the original number of features is greater than 30. When the starting number of features is smaller than 19, the ensemble based only in the RS method provides better results. Cunningham and Carney try to explain why this difference in performance. They say that when the initial number of features is greater than 30, there may be less risk of loss diversity in searching for good quality ensemble members (Cunningham and Carney, 2000). In the

other hand, when the starting number of features is smaller than 19, creation of base classifiers has to take into account diversity and accuracy (Cunningham and Carney, 2000).

Opitz mentions that *ensemble feature selection* is a harder than traditional feature selection because it is required not only to find features relevant for the particular task and the learning algorithm, but also find feature subsets that will promote disagreement among ensemble's classifiers (Opitz, 1999). His algorithm called GEFS (Genetic Ensemble Feature Selection) uses GA to search feature subsets that make neural network classifiers accurate and diverse in their predictions (Opitz, 1999). The search of GA considers the accuracy and diversity by including them in the fitness function. As the author mentions, surprisingly the initial population in GA is already good, with performance comparable to popular ensemble creation techniques as bagging and boosting (in particular AdaBoost). He remarks that the lost in performance by having individuals using only a subset of features (instead of all set of available features) is compensated with the diversity created between classifiers (Opitz, 1999). In (Tsymbal *et al.,* 2005), diversity is also an element considered in ensemble feature selection. They compared four search strategies together with RS: genetic search, hill-climbing, ensemble forward and backward sequential selection. They concluded that diversity measure used influences the ensemble feature selection process and results differ depending on the context of use of diversity and on the data type being processed (Tsymbal *et al.,* 2005).

3.4.3 Ensemble of classifiers and genetic programming

Re-sampling techniques derived from different ensemble methods have been also applied to GP in order to improve its capabilities. Iba has applied bagging and boosting to GP to create ensembles as described in (Iba, 1999): the whole GP population is divided in to a set of populations and each population evolves independently and the best individuals for each subpopulation vote to form a composite tree output which will be applied to the test data (Iba, 1999). He applies this method to three problems: regression, time series and price prediction in the financial market. In the three cases, average performance is improved for

ensembles of GP-based classifiers. He shows that solution trees are also smaller than those produced by standard GP, so they assure that this is due to the re-sampling techniques used to create the ensembles. Unfortunately there is no specific information about the way in which the vote from each subpopulation is combined for the final answer so that the comprehension of the effect on tress sizes would be clear.

Zhang and Bhattacharyya used GP to build base classifiers and construct an ensemble to classify large-scale data of about 300.000 samples (Zhang and Bhattacharyya, 2004). Base classifiers are trained with small data sets randomly selected with replacement and the ensemble decision is taking using majority vote. The superiority of the GP-ensemble is partly attributed by the authors, to the diversity among base classifiers, in terms of functional form and raw characteristics used in each classifier (Zhang and Bhattacharyya, 2004).

Folino *et al.* proposed a modified version of bagging in which each base classifier is trained on a different segment of the overall data and then they are combined to classify new data by applying simple majority vote (Folino *et al.*, 2003). Training base classifiers on different segments (with a small number of samples compared to the original size of the data set) produces different classifiers. As they state in (Folino *et al.*, 2003), the main advantage of their approach is to obtain comparable accuracies to a single classifier (trained with the whole data set), but at a much lower computational cost. The classifiers of the ensemble are built by applying Cellular GP. In this method, each subpopulation generates a classifiers working on a segment of the training set (Folino *et al.*, 2003). In addition, there is sporadic exchange of information between evolving subpopulations, represented by the better individuals. This exchange reduces the size of the individuals (or classifiers), and consequently the time to evaluate the fitness is also reduced (Folino *et al.*, 2003).

As mentioned before, diversity amongst base classifiers is important to generate successful ensembles. Hong and Cho compare the structure of the classification rules to build ensembles with diverse classifiers (Hong and Cho, 2006). The classification rules are generated using GP and then the diversity between them is estimated with an edited distance. The set of five

classification rules which maximizes diversity are selected to build the final ensemble. This method is applied to classification of cancer based on DNA micro-array data and is compared to ensembles created by neural networks showing the improvement due to the diversity measure (Hong and Cho, 2006).

In the cases already presented, GP was used to generate base classifiers and then the ensemble was built by combining the GP-based classifiers. Different strategies were used to choose diverse classifiers and improve the accuracy on unseen data. In the next chapter, we present the implemented method to generate GP-based classifiers which will be used in the ensemble creation in subsequent chapters.

CHAPTER 4

FEATURE CREATION FOR CLASSIFIERS WITH GENETIC PROGRAMMING

Previous chapters presented an overview of the theoretical basis that will be used in this chapter to develop a method that constructs features for classifiers. In Chapter 2, we have shown that feature subset selection methods improve classifier performance, by eliminating redundant features and reducing the number of features used for classification. GA have been very successful in feature subset selection problems (Kudo and Sklansky, 2000; Ferri *et al.,* 1994; Jain and Zongker, 1997; Oliveira *et al.,* 2003a). The concept of feature selection can be extended for ensembles, so that base classifiers have feature sets that maximize the ensemble performance (Guerra-Salcedo and Whitley, 1999a; Guerra-Salcedo and Whitley, 1999b, Oliveira *et al.,* 2003c). In the case of feature subset selection, the resultant classifiers are optimized and then they are used as constituents of an ensemble with the hope that the combination generates a performing ensemble. Optimization of classifiers can be done in a different way, by constructing a set of features (from an initial set), instead of selecting features. In Chapter 2, we mentioned the example of body mass index that shows the advantages that GP has in comparison to GA when constructing features. Once individual classifiers have been optimized, they can be combined to generate a performing ensemble.

From the feature construction methods described in Chapter 2, the algorithm proposed in (Bot, 2001) ahs been selected. The purpose of this algorithm is to automatically create evolved features from an original set of features with the express aim of dimension reduction and the additional aim of improving the accuracy of a classifier (Bot, 2001). Bot applied his method to two classifiers: Minimum Distance to Means (MDM) and *k*-NN. MDM is a weak classifier that does not require training and assigns the class for unseen data which its mean is closer to the tested example. GP is used because they are able to establish the structure and values of the solution and, in contrast, GA is tied to a specific predefined structure of the solution (Guo *et al.,* 2005). Bot tested his method with 16 datasets from UCI repository (Newman *et al.,* 1998). Fitness measure used in Bot's method is based on the recognition rate

of the classifiers (*k*-NN and MDM). We also propose another fitness function that maximizes the inter-class scatter over the intra-class scatter using the Fischer criterion (Duda *et al.*, 2001; Guo *et al.*, 2005).

This chapter is organized as follows: Section 4.1 presents the main structure of the GP-based feature construction algorithm. We explain how the algorithm works considering the same parameter settings used in (Bot, 2001). Further analysis and improvements are presented in chapter 5. In this chapter, we develop an experimental protocol to verify the quality of the solutions generated by Bot's method with all 16 datasets used in (Bot, 2001). The GP tool called Open BEAGLE (Gagné and Parizeau, 2004a; Gagné and Parizeau, 2004b) was used to develop feature construction algorithm. This tool is briefly described in Section 4.2. Section 4.3 presents the options used to quantify the fitness function that guides the GP evolution process: *k*-NN, MDM and fitness based on Fisher criterion. The experimental protocol is developed in Section 4.4. We include a description of data sets, a general description of the experiments, the parameter settings required, the results and their analysis, as well as some interpretation of the results. Finally, in Section 4.5, a discussion about the main topics developed in this chapter is presented.

4.1 Feature construction with Genetic Programming – Bot's algorithm

The purpose of the algorithm proposed in (Bot, 2001) is to automatically create evolved features from an original set of features with the express aim of dimensionality reduction and the additional aim of improving the accuracy of a classifier (Bot, 2001). The automatic construction of evolved features is done by using GP as search strategy. Each of the evolved features is created at a time. Initially, a standard GP is run for *Nbr_Gen* generations and the best individual from this population is selected and becomes the first evolved feature, noted as EF(f=1). In this notation, EF(f) means the mathematical expression of evolved feature f. Creation of subsequent features considers the jointly performance of previous evolved features and selected individual for the current feature, to determine the improvement in performance that new features offer. This process of creation of evolved features continues

until Bot's stopping criterion is met or a pre-defined maximum number of features *Max_Nb_features* is reached. Bot's stopping criterion detects the moment when additional evolved features do not engender a significant improvement in performance.

Bot's algorithm introduces a Bot_stopping criterion which identifies when to stop adding new evolved features. As Bot's method is implemented with GP, we have to consider as well the classical stopping criterion, referred here as Evol_stop frequently used in evolutionary methods. Evol_stop is defined in terms of an acceptable error rate (or recognition rate) level or a pre-defined number of generations. In practice, the second criterion is more often used because expected error rate is normally not attained within a reasonable number of generations. As a result, GP is run for a large number of generations which means a long time. Bot's method (Bot, 2001) tries to avoid the long and time-consuming evolution by creating one feature at a time and guiding the evolution of new evolved features with the aid of the improvement in recognition rate of the proposed new feature in conjunction with the already evolved features. This method generates solutions with small number of features, with acceptable recognition rates and taking as few as 10 generations (Bot, 2001). The general structure of the algorithm, the calculation of the fitness function and the stopping criterion are described in the following subsections.

4.1.1 Evolution framework

From an initial set of raw features, GP constructs features to improve performance of a classifier, that is, to optimize the original classifier built with the RS method. Construction of evolved features is done one at a time and the process continues until the addition of new evolved features do not produce a considerable increment in performance. The maximum number of evolved features, *Max_Nb_features,* is an input parameter that depends on each specific application. The general algorithm is presented in the following page (see Algorithm 2). To create first evolved feature, noted as EF(f=1), a standard GP is run for *Nbr_Gen* generations and the best individual from the last population P(f=1, g= *Nbr_Gen*) is selected and becomes EF(f=1). To explain the procedure for evolved features greater than one,

suppose that i-1 "definite evolved features" have been already defined, that is EF(1), EF(2), ..., EF(i-1), with ($1 \leq i\text{-}1 \leq Max\_Nb\_features\text{-}1$). GP is run for *Nbr_Gen* generations to create the i evolved feature EF(i).

Algorithm 2: Bot's algorithm

Input:
Nbr_Gen : Number of generations to evolve solutions for one feature
Max_Nb_features : Maximum number of evolved features
Output:
Def_Sol : Definitive selected individual (formula) for each feature

start - Bot's algorithm

```
1  f=1; EF(f=1)=Φ; g =0;                    // EF defined as unfilled
2  Initialize population P(f,g)
3  Evaluate fitness(P(f,g))
4  Run_Evol (f=1, g>0)
5  F(f=1) = Best_Indv(P(f))                 // Selected_Indv(f=0)

6  repeat
7     EFprior(f-1) = EF(f-1) ⊕ EF(f-2) ⊕ ... ⊕ EF(1)
8     g =0
9     Initialize population P(f,g)
10    Evaluate fitness(P(f,g) , EFprior(f-1) )
11    Run_Evol ( f>1, g>0)
12    Selected_Indv(f) = Best_Indv(P(f))          // Selected_Indv(f)

13    stop = Bot_stopping(Selected_Indv(f), EFprior(f-1), f)
14    if ~stop
15       EF(f) = Selected_Indv(f)
16       EFprior(f) = EF(f) ⊕ EFprior(f-1)       // new features to de added
17       f = f+1
18    else
19       Def_Sol = EFprior(f-1)= EF(f-1) ⊕ EF(f-2) ⊕ ... ⊕ EF(1)   // no more features
20    end-if
21 until stop criterion met
```

end – Bot procedure

Fitness calculation (see details in Section 4.1.2) for each individual in population P(f=i, g) considers the individual of the current run juxtaposed to already evolved features EF(1), EF(2), ..., EF(f=i-1). The juxtaposition is $EF_{prior}(f\text{-}1) = EF(1) \oplus EF(2) \oplus \ldots \oplus EF(f=i\text{-}1)$.

The best individual is called Selected_Indv($n=i+1$, g) and the Bot's stopping criterion determines if selected individual is definitely accepted to become EF($f=i$) or not. The selected individual is accepted when its contribution is higher than a threshold (see Section 4.1.3). In this case, the set of definite evolved features is EF(1), EF(2), ..., EF($f=i$) and new evolved features can be added. When the selected individual is not accepted, evolution is stopped. That case means that it is no valuable the increment in performance that the selected individual represents and so, it is discarded. The set of definite evolved features remains EF(1), EF(2), ..., EF(i-1) becomes the final solution and no more features are added. In Algorithm2, the step called Run_Evol (f, g>0) corresponds to a standard running of GP for any feature value and generations after the initial one. Fitness evaluation within it depends on the number of already evolved features. The steps (4 and 11) included within Run_Evol (f, g>0) are detailed in the full version of Bot's algorithm presented in Annex II.

4.1.2 Fitness function

To explain the procedure for fitness calculation for evolved features, suppose again that i-1 "definite evolved features" have been already defined, that is EF(1), EF(2), ..., EF($f=i$-1), with ($1 \leq i\text{-}1 \leq Max\_Nb\_features\text{-}1$). During the running to create the i-th evolved feature, each individual in population P($f= i$, g) can be seen as a tree or formula expressed in terms of the mathematical operators and raw data. Fitness measure is calculated as follows: the formula for each individual in population is evaluated (using raw data values). When replaced raw data values and executed mathematical operations in the formula, a number is obtained. After applying the formula to all samples in optimization database, a vector is formed with all values obtained. The definitive evolved features EF(1), EF(2), ..., EF($f=i$-1) are considered to calculate the fitness of the each individual. As they are already defined, they are (i-1) vectors. In total, there are i column vectors for each individual in the population. These i column vectors are the input to a classifier that determines the corresponding recognition rate generated and this is the fitness value assigned to the individual considered. The same process presented above is repeated for every individual (formula) in the population P($n= i+1$, g) for each generation, only changing the last column

vector (the *i*-th) that is, the interpretation of the individuals considered. At the end of *Nbr_Gen* generations, the best individual (highest fitness value) is selected. This individual is called Selected_Indv(*f=i, g= Nbr_Gen*). To decide if the selected individual becomes a definite evolved feature (EF(*i*)), the Bot's stopping criterion (details in next section) is evaluated.

Bot used two different classifiers: MDM and *k*-NN (Bot, 2001). MDM is proposed in addition to the k-NN, because of its computational simplicity (Bot, 2001). MDM classifier calculates the mean of each class over all current evolved attributes at the present evolution step *i* and assigns the class which mean is closest to the evaluated sample (Bot, 2001).

4.1.3 Bot's stopping criterion

Evolution is done as explained in the previous section. Bot proposed a stopping criterion based on the relative contribution of each new feature to the recognition rate (Bot, 2001) and its role is to determine when is not acceptable to have an additional evolved feature for the small increment in performance that this new feature represents (trade-off between performance increment and an additional evolved feature) Then, this criterion indicates the stage at which no more features are needed.

To explain Bot's stopping criterion we suppose that *i*-1 "definite evolved features" have been already defined, that is EF(1), EF(2), …, EF(*f=i*-1), with ($1 \leq i\text{-}1 \leq Max\_Nb\_features\text{-}1$), GP has been run for *Nbr_Gen* generations and the selected individual Selected_Indv(*f=i, g= Nbr_Gen*) has been chosen. If recognition rate –calculated as explained the previous section– is smaller that a constant value *d,* the Selected_Indv(*n=i, g= Nbr_Gen*) is taken away and the process stops. If recognition rate is higher than *d*, the Selected_Indv(*n=i, g= Nbr_Gen*) contributes to the objective and is included in the definite evolved feature set (that has now *i* features): EF(1), EF(2), …, EF(*f=i*-1), EF(*f=i*) . The process continues until recognition rate is less than the constant *d*.

In each experiment, d is calculated as the relative incremental recognition rate on the optimization database for the evolved feature $f_{\max\_val}$ where recognition rate on the validation dataset $v(f)$ is maximal. Therefore, $f_{\max\_val}$ is:

$$f_{\max\_val} = \arg\max_{f}\{v(f)\} \tag{4.1}$$

and the threshold results:

$$d = \frac{t(f_{\max\_val}) - t(f_{\max\_val} - 1)}{t(f_{\max\_val} - 1)}. \tag{4.2}$$

where t($f_{\max\_val}$) is the optimization recognition rate for feature $f_{\max\_val}$.

4.1.4 Block diagram

Figure 4.1 shows the block diagram of Bot's method. Input data is a collection of examples in an N-dimensional space according to the number of attributes of the data (we do not make reference here to any sampling method to built classifiers as in Chapter III). Attributes of input data correspond to the terminal set used by the GP algorithm. With this input, GP searches to construct an evolved classifier. So, GP is used to optimize the classifiers. That is, the creation of evolved features with GP is the optimization of the classifier. As a result, the input data to the GP algorithm is called the *optimization* dataset. GP as other evolutionary algorithms, performs a guided search by qualifying each of the proposed solution (individuals in each generation), so that we consider this framework as a *wrapper* approach (Duda *et al.*, 2001). This is the purpose of the fitness function. Qualification of each solution is done by a classifier algorithm. In this case, a k-NN or a MDM are used. These classifiers have to use a different dataset to avoid a biased qualification. In general this set is called *training* dataset.

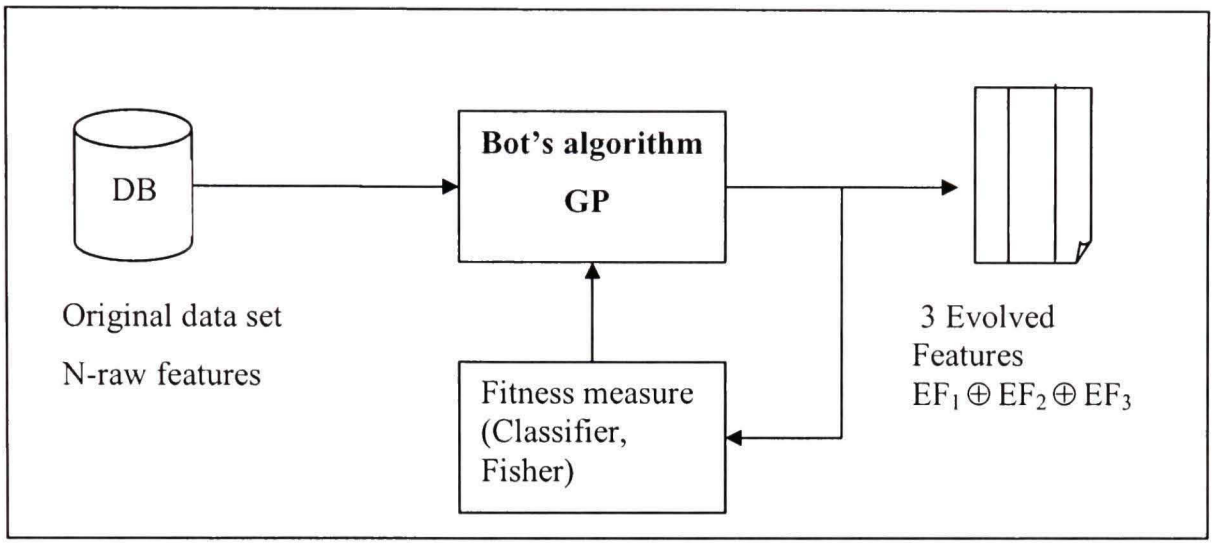


Figure 4.1 ***Block diagram of Bot's method.***

GP programming used an optimization dataset to search for evolved classifiers. Fitness is measured by a classifier (k-NN or MDM) which used a training dataset. Validation dataset corroborates the solution found during optimization and a test data set measures the generalisation power of the validated solution.

These classifiers have to use a different dataset to avoid a biased qualification. In general this set is called *training* dataset.

4.2 Genetic programming tool: Open BEAGLE

As described in its website (http://beagle.gel.ulaval.ca/): "Open BEAGLE is a C++ Evolutionary Computation (EC) framework. It provides a high-level software environment to do any kind of EC, with support for tree-based GP; bit string, integer-valued vector, and real-valued vector GA; and evolution strategy". Our purpose in this section is to present a succinct overview the Open BEAGLE that enables the reader to have the "big picture" of it. This is a complex tool and a more ample description can be found in (Gagné and Parizeau, 2004a; Gagné and Parizeau, 2004b) and its website.

4.2.1 Open BEAGLE architecture

The framework architecture of Open BEAGLE follows Object Oriented programming principles (Gagné and Parizeau, 2004a; Gagné and Parizeau, 2004b). Its architecture is

divided in three levels. The first level, named OO foundations, can be considered as an extension of C++ and the Standard Template Library (STL) (Gagné and Parizeau, 2004b), as shown in Figure 4.2 (taken from (Gagné and Parizeau, 2004b)). The second level, implements the generic framework for Evolutionary Computations (EC). In the third level, different EC frameworks are developed as modules (for GA, GP and Co-evolution) (Gagné and Parizeau, 2004b).

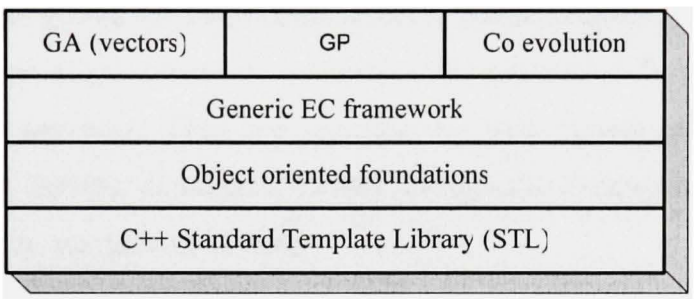


Figure 4.2 ***Architecture of Open BEAGLE framework.***
(taken from Gagné and Parizeau, 2004b)

4.2.2 Object oriented foundations

Classes in Open BEAGLE are derived from an abstract class called Object. Open BEAGLE follows inheritance principles, then objects are dynamically instantiated (Gagné and Parizeau, 2004a; Gagné and Parizeau, 2004b). Allocators are object factories that can assign, clone, and copy specific types of objects Gagné and Parizeau, 2004a; Gagné and Parizeau, 2004b). Containers are dynamic arrays of object pointers that use allocators to instantiate the objects that it has. In addition, information about the population, like parameter values and evolution results can be written in XML format, which can be visualized in a web browser Gagné and Parizeau, 2004a; Gagné and Parizeau, 2004b).

4.2.3 Generic EC framework

It is composed of a generic structure of populations, an evolution system, and a set of operators includes in an evolver (Gagné and Parizeau, 2004b). Population is arranged in four hierarchical levels: *vivarium, demes, individuals and genotypes*, as presented in Figure 4.4 (taken from (Gagné and Parizeau, 2004b)). The *vivarium* includes the individuals present in the evolutionary system. It presents statistics on the population. A *deme* is a close environment where a group of individuals evolve independently (Gagné and Parizeau, 2004b). Individuals can migrate between demes at each generation. *Individuals* represent the proposed solution at any time. They are specified by their fitness measure and genotype (Gagné and Parizeau, 2004b). *Genotypes* contain the genetic description of each individual (for instance in GP they are defined as trees).

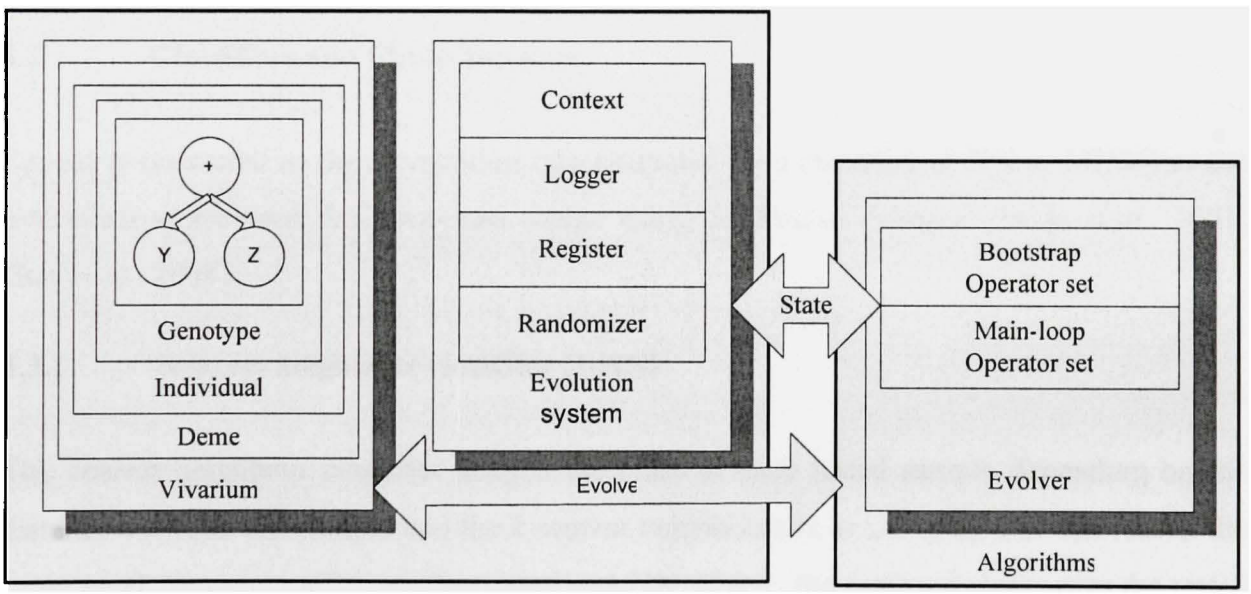


Figure 4.3 ***Open BEAGLE generic EC framework.***
(taken from (Gagné and Parizeau, 2004b))

As shown in Figure 4.3, the framework also includes an Evolution system. In turns, it is composed of *context allocator, register, logger and randomizer.* The context allocator presents the state of the evolution process. *Register* is used to centralize information about different parameters (which is distributed within different elements). The *logger* process all

messages generated by EC framework. It acts as a user interface. The *randomizer* generates random numbers (Gagné and Parizeau, 2004b).

4.2.4 General operation of EC framework

The evolving process (see right hand side of Figure 4.3) consists of a sequence of operations iteratively applied to demes of the vivarium. Each genetic operation is defined as an operator. The evolver has two operator sets (Gagné and Parizeau, 2004a; Gagné and Parizeau, 2004b): bootstrap and the main-loop operators. In the bootstrap operator, the series of operations applied to each deme to construct the initial population are listed. The main-loop operator presents the list of iterative operations that are performed to the population after the initial population.

4.3 Classifiers and fitness function

Fitness is measured as the recognition rate estimated by a classifier (k-NN or MDM) or the inter-class scatter over the intra-class scatter using the Fischer criterion (Duda *et al.*, 2001; Guo *et al.*, 2005).

4.3.1 Nearest neighbour classifier (k-NN)

The nearest neighbour classifier assigns the class to each tested sample depending on the distance between this sample and the k nearest neighbours ($k \in \mathcal{N} \geq 1$). For this reason the nearest neighbour classifier is often noted as k-NN. If k=1, the assigned class set to the tested sample is the class of the closest sample on the training set. If k >1, the classifier ranks the k nearest neighbours and counts the votes for each class between the k-selected neighbours. The class with majority vote is assigned to the tested sample. As k increases, decision boundaries between classes are smoother as decision is taken over a higher number of neighbours.

4.3.2 Minimal distance to means (MDM)

This classifier assigns the class to each tested sample depending on the distance between this sample and the classes' means calculated over the training data set. The class, whose mean is closest to the tested sample, is assigned as class for the tested sample. This classifier brings two main advantages: it does not require training apart from the calculation of mean for each class. Another advantage is the simplicity in computation to get the answer: only the distance from the tested sample to each class mean. The increment in computation is linear with the number of tested samples (Bot, 2001). In the other hand, it is known that this classifier does not provide an excellent recognition rate but this force the GP to search for better individuals (Bot, 2001).

4.3.3 Fisher Linear Discriminant Analysis

In Section 2.5 we outlined the method used in (Guo *et al.*, 2005). In this case, GP generates mathematical expressions guided by a fitness function that maximizes the inter-class scatter over the intra-class scatter using the Fischer criterion (Guo *et al.*, 2005; Duda *et al.*, 2001). By the use of Fisher criterion to express the fitness function, GP tries to maximize the difference between two-classes in an iterative process applied to pairs of classes (Guo *et al.*, 2005). The fitness function measure for each pair of classes i, j based on the Fisher criterion can be expressed as (Guo *et al.*, 2005):

$$fit_{i,j} = \frac{\left| \frac{1}{n}\sum_{i=1}^{n} S_i - \frac{1}{m}\sum_{j=1}^{m} S_j \right|}{\sqrt{\frac{\sum_{i=1}^{n}\left(S_i - \frac{1}{n}\sum_{i=1}^{n} S_i \right)^2}{n-1} + \frac{\sum_{j=1}^{m}\left(S_j - \frac{1}{m}\sum_{j=1}^{m} S_j \right)^2}{m-1}}} \tag{4.3}$$

In equation 4.3, fit_{ij} is the fitness function measure between classes i and j, S represents the examples of the data set, the numerator reflects the distance between classes i and j and the denominator denotes the range of variance within classes i and j. In the case of a number of classes greater than 2, the Fisher criterion is decomposed in $q=C(K,2)=K(K-1)/2$ two-class Fisher criterion calculations.

4.4 Experimental Protocol for Bot's method

Bot applied his method to 16 data sets from UCI repository (Newman *et al.*, 1998). We replicate the experiments to verify the results concerning performance and stopping criterion. In the experiments we followed the settings as established in (Bot, 2001) as much as possible.

In the first subsection we briefly describe the data sets used, the normalization made to the data and how the available data is partitioned into the four disjoint blocks for optimization, training, validation and test. In the second subsection, we describe the GP tool used to run our experiments. Following subsection describes the overall system applied and the experiments with different classifiers and fitness measures. Following subsection presents the results and makes an analysis and a comparison to Bot's results. It includes comments about the application of global validation procedure.

4.4.1 Database description and usage

Databases used in (Bot, 2001) are part of a Machine Learning database repository from the University of California at Irvine (Newman *et al.*, 1998). Most of the data sets have two files with the following format: "db_name.data" and "db_name.name", where "db_name" refers to the name of each specific data set and the extensions ".data" means that this file contains the data and ".name" contains a description of the data set (number of instances, number of classes, type of data contained and a brief description of the problem). From the 16 data sets

used by Bot, he assures that only databases with no missing values and with numeric values were selected (Bot, 2001). Notwithstanding, some of the data do not fulfill these conditions. For instance, data set "car" has categorical values and "cmc" contains binary data. In addition, Bot took a random subset from "waves" data set. As a result, not all data sets used in our experiments correspond 100% to the data used by Bot. Following is a list of the databases used and their main characteristics (see Table 4.1): number of samples, raw features and classes. The amount of instances from different classes is not always well balanced in some of the data sets, so the percentage of the main class is indicated. When no number appears in this column, classes are evenly distributed or in similar percentages.

Table 4.1

Characteristics of databases used by (Bot, 2001)

Database	Number of cases	Number of variables	Number of classes	Percentage majority class (%)
Australian	690	14	2	
Bupa	345	6	2	
Car	1728	5	4	70
Cmc	1473	9	3	
Ecoli	336	7	8	43
German	1000	24	2	
Glass	214	9	16	36
Ionosphere	351	34	2	64
Iris	150	4	3	
Pima	768	8	2	65
Segmentation	2310	19	7	
Sonar	198	60	2	53
Teaching	151	5	3	
Waves	330	21	3	
Wine	178	13	3	40
Yeast	1484	8	10	31

Data is normalized to transform all values to the range [0, 1]. Normalization is made for each attribute by using Min-Max method. Values v(i) in the range [minA, maxA] are transformed according to equation 4.4 to a new range [new_minA, new_maxA]:

$$\hat{v}(i) = \frac{v(i) - \min A}{\max A - \min A} * (new\_\max A - new\_\min A) + new\_\min A, \tag{4.4}$$

Usually the range [new_minA, new_maxA] is [0, 1]. In this case the equation is simplified to:

$$\hat{v}(i) = \frac{v(i) - \min A}{\max A - \min A} \tag{4.5}$$

As shown in Table 4.1, the maximum number of instances is 2310 for the segmentation data set. As the size of data sets is not big, we use 10-fold cross-validation. So, data is partitioned in 10 non-overlapping blocks called folds of approximately the same size. One block for optimization, one block for validation, another block for test and the remaining 7 blocks for training. Since some data sets have an inherent unbalance in the number of samples for different classes, we tried to replicate the same class distribution as in the original data set, but this is not always possible. For instance, Ecoli data set has 336 samples with 8 classes. One class has 143 samples (approximately 42% of the whole data set), meanwhile two other classes have each one 2 samples. Therefore, when splitting data into folds, there will be training folds without any samples from these two classes and test or validation folds that contain one or the two samples from these classes. This imbalance of class distribution produces a decrement in the performance of some folds that could affect the final results. For these cases, the samples for each fold are randomly selected.

4.4.2 Description of experiments

Bot applied his algorithm to 16 data sets from UCI repository (Bot, 2001). The main purpose here is to replicate the experiments to verify the Bot's method. As a result, the effectiveness of using two or three evolved features to represent a data set, instead of the whole set of raw characteristics, can be established. Parameters for the experiments are set according to (Bot, 2001) when the information is available. When a parameter value is not specified in (Bot, 2001), we set it to the default value defined in the GP tool used.

The general diagram of the system was shown in Figure 4.1. The system uses GP to create features called *evolved features*, so that they would be useful to classify unseen data of the problem. Optimization data is the input to the GP algorithm. Guided by the fitness function, it searches for solutions that produce fittest individuals. Fitness is measured as the recognition rate estimated by a classifier (k-NN or MDM) or the inter-class scatter over the intra-class scatter using the Fisher criterion (Duda *et al.*, 2001; Guo *et al.,* 2005). In both cases, two data sets are used: the training and the optimization data sets. Interpreting the evolved individuals with the training data generates the prototypes for the k-NN classifier (proximity of neighbours) or the classes' mean for MDM classifier and the mean and variance used for Fisher criterion. Through the evolution, for every generation and evolved feature, all individuals in the population are interpreted according to the optimization data set and this is the input to a classifier which assigns the class to each sample of the optimization data set. During the search for an evolved classifier, the overall system can generate solutions that over fit the dataset used for this search (*optimization* dataset) if an adequate validation method is not used. Hence, another dataset is used to corroborate (validate) the solutions proposed by the framework GP-(k-NN or MDM classifier). This set is called *validation* dataset. The available data set is divided in ten disjoint folds which are used as follows: one fold for optimization, a second fold for validation, and the remaining eight folds are used for training. This process is repeated 10 times so that optimization and validation are applied to different folds each time. Final recognition rate is calculated as the average over all folds and replications.

As presented in Figure 4.1, there are two main elements in the GP-based system: the classifier and the GP algorithm. The parameters of the two components of the GP-classifier wrapper have to be defined before the optimization phase.

4.4.2.1 Parameters related to classifier or fitness measure

Two classifiers are used to measure the fitness of individuals: *k*-NN (*k*-NN) and minimal distance to means (MDM).

(i) Nearest Neighbour (k-NN)

Table 4.2 indicates the settings for number of neighbours and prototypes used.

Table 4.2

Parameters of k-NN classifier

Parameter	*Interpretation of the parameter*	*Value set*
k: Number of neighbours	Number *k* of neighbours considered to take the decision about the class of the tested sample	$k = 3$
Number of prototypes	Number of sample (prototypes) considered when calculating distances	Seven folds in 10-fold cross-validation case (explanation in Section 4.3.1)

(ii) Minimal Distance to Means (MDM)

It does not require a definition of parameters.

(iii) Fitness measure based on Fisher criterion

The fitness function measure for each pair of classes *i, j* based on the Fisher criterion was specified in equation (4.3). This equation is applicable for two-class problems. When the number of classes is greater than 2, the Fisher criterion is decomposed in *q=C(K,2)=K(K-1)/2* two-class Fisher criterion calculations. Fitness measure is oriented towards the worst case or the minimum value of all two-class calculations, so the application of this value to other two-class pairs produces a good classification. The final fitness measure fit_{Fish} is defined in (Guo *et al.*, 2005) as:

$$fit_{Fish} = \frac{1}{\beta}\left(\min(fit) + \lambda.\frac{1}{q}\sum_{i=1}^{q}(fit)\right), \tag{4.6}$$

where *fit* is an array of all two-class Fisher criteria among classes. Parameter λ is empirically set to 0.001 (Guo *et al.*, 2005), is a factor that considers the contribution from the mean value. For instance, if the minimum values for two features are similar, the one with the largest average of values will survive. Finally, a value β is experimentally applied to re-scale the resulting fit_{Fish} values less than 100 (the maximum fitness value set during evolution). Value of β scale is adjusted according to each data set. See further details about these parameters in Section 5.3.2.

4.4.2.2 GP-related parameters

Before running a GP program, there are some steps that have to be done (Koza, 2004; Koza, 1992):

(1) Specification of the set of terminals;
(2) Specification of the set of functions;
(3) Definition of fitness measure;
(4) Specification of parameters to control the program run (i.e., mutation and cross-over probabilities for individuals, size of the population, etc);
(5) Termination criterion (normally expressed as maximum number of generations or problem-resolution success measure).

The mentioned steps are explained and then their values are presented in Table 4.3 above. Based on Bot's method (Bot, 2001), we use a set of functions composed by the standard mathematical operations addition, subtraction, multiplication, protected division. Protected division avoids problems when dividing by zero, defining it as one. The terminal set is composed by the number of attributes o raw features for each specific data set; for instance the data set called Australian has 14 raw features, so the terminal set would be: { x_1,..., x_{14}, ephemeral random constant }, ephemeral random constant is a random floating point constant ranging from -1.0 to 1.0. Fitness measure is taken as the recognition rate of a *k*-NN classifier (or a MDM classifier) measured over the optimization data set.

The main parameters to control the GP are: population size, number of generations, selection mechanism, cross-over and mutation probabilities, the method to create the initial population, the maximum tree depth and if elitism is activated or not. For the experiments with the 16 data sets used in (Bot, 2001), we have chosen the values indicated in (Bot, 2001) as shown in Table 4.3 above. It is worth to include an explanation about two other parameters: maximum tree depth and creation of initial population. Maximum tree depth refers to the allowed number of levels that any individual (tree) in the population can have at maximum. This parameter tries to control the occurrence of extremely big trees, because operations applied to them (interpretation, cross-over, etc.) increase the computation time. Maximum tree depth depends on the difficulty of the problem (Koza, 1992). Smaller tree depth value means shorter trees and possible easier interpretation. Two additional parameters are related: *gp.init.mindepth* and *gp.init.maxdepth* (see Table 4.4 above). They are the minimal and maximum tree depths when trees are created. A standard maximum tree depth value of 17 has been used to solve a wide type of problems, as presented in (Koza, 1992), was also used in (Bot, 2001) and set here at that value. Minimal and maximum values for initial trees have been defined (as indicated in Table 4.4 above) in 2 and 5 respectively.

The second parameter, creation of initial population, is defined as ramped half-and-half (see Table 4.3). This means that an equal number of trees of different depths are produced and, for each depth, value, 50% of the trees are created via the "full method" and 50% via the "grow method" (Koza, 1992). The "full method" to generate the initial random population creates trees with each path with a length equal to the specified depth (Koza, 1992) (path goes from root to the end-point). The "grow method" creates trees of various shapes with lengths of each path no greater than the specified length (Koza, 1992). In our case, an equal number of trees with depths between 2 to 5 are created, that is, approximately 25 individuals (25% of the population) are created of size 2, 25 individuals of size 3, and so on, until 25 individuals of size 5. From the 25 individuals of each depth, approximately half are created with the "full method" and half with the "grow method". The number 25 comes from the population size divided by the different number of depths, which approximately is 100/4.

Table 4.3

GP related parameters

Objective	To find out the minimal number of mathematical expressions that maximize recognition rate on optimization data set (Table 4.1)	
Terminal set	{Variables within the set ($x_1,\ldots,x_q,\ldots,x_{N\_ATTR}$), ephemeral random constant } Variables x_q are the raw features from each data set. N_ATTR is the maximum number of raw features (attributes) in each data set. Variables are normalized according to equation (4.5). Ephemeral random constant is a random floating point constant ranging from -1.0 to 1.0	
Function set	{+, -, *, / }. Division (/) refers to protected division to avoid division by zero	
Fitness measure	A classifier that maps the last evolved mathematical expression in conjunction with previous evolved mathematical expressions into any of the dataset classes by using the optimization data set.	
Parameters	Population size	100
	Number of generations	11
	Selection	Tournament selection 7 individuals
	Cross-over probability (individual)	0.9
	Mutation probability (individual)	0.1
	Creation Initial population	Ramped half-and-half
	Maximum tree depth	17
	Replacement mechanism	Steady –state
	Elitism	Keep one individual
Success predicate	Individual tree representation with 100% classification rate (over validation data set)	

Finally, the replacement strategy chosen in (Bot, 2001) is steady-state. In this replacement mechanism, the whole population is not necessarily replaced at each generation as it happens in the conventional GA (also applicable to GP) replacement strategy called generational. Steady-state method imitates what happens with many animal species in which parents and children can live at the same time with not defined boundaries (Langdon, 1998). This means that off spring once created they are added immediately to the population and are available for reproduction (Beasley et al., 1993). The implementation of steady-state in Open BEAGLE replaces one individual at a time and immediately adds the off spring into the population. The system considers a generation the moment when a number of individuals

equal to the population size has been replaced. Additional information on steady state replacement included in Section 2.2.3.4 and in Annex I.

Table 4.4 shows the definition and values set for additional parameters required for a GP run. When there is no specific criterion to set an initial value, the default value used in Open BEAGLE is taken.

Table 4.4

Additional parameters to define for a GP run

Parameter	Explanation	Initial value
Cross-over prob. *gp.cx.indpb*	Individual crossover probability at each generation	(def: 0.9)
gp.cx.distrpb	Probability that a crossover point is a branch (node with sub-trees). Value of 1.0 means that all crossover points are branches, and value of 0.0 means that all crossover points are leaves.	(def: 0.9)
Mutation prob. *gp.mutstd.indpb*	Standard mutation probability for an individual. A standard mutation replaces a sub-tree with a randomly generated one.	(def: 0.05)
gp.mutshrink.indpb	Shrink mutation probability for an individual. Shrink mutation consists in replacing a branch (a node with one or more arguments) with one of his child node. This erases the chosen node and the other child nodes	(def: 0.05)
gp.mutswap.distrpb	Probability that a swap mutation point is a branch (node with sub-trees). Value of 1.0 means that all swap mutation points are branches, and value of 0.0 means that all swap mutation points are leaves. Swap mutation consists in exchanging the primitive associated to a node by one having the same number of arguments.	(def: 0.5)
gp.mutswap.indpb	Swap mutation probability for an individual. Swap mutation consists in exchanging the primitive associated to a node by one having the same number of arguments.	(def: 0.05)
gp.mutstd.maxdepth	Maximum depth for standard mutation. A standard mutation replaces a sub-tree with a randomly generated one.	(def: 5)
gp.init.maxdepth	Maximum depth for newly initialized trees.	(def: 5)
gp.init.mindepth	Minimum depth for newly initialized trees.	(def: 2)
gp.try	Maximum number of attempts to modify a GP tree in a genetic operation. As there are topological constraints on GP trees (i.e. tree depth limit), it is often necessary to try a genetic operation several times.	(def: 2)
ec.repro.prob	Probability than an individual is reproduced as is, without modification. This parameter is useful only in selection and initialization operators that are composing a breeder tree	(def: 0.1)

4.4.3 Results

Experiments were run according to the experimental protocol detailed in preceding sections. Two different approaches were used to calculate the fitness measure: recognition rate of a classifier (*k*-NN and MDM) and the inter-class scatter over the intra-class scatter using the Fischer criterion (Duda *et al.*, 2001; Guo *et al.*, 2005). We present the results of our implementation of Bot's method using Open BEAGLE (Gagné and Parizeau, 2004b) as GP tool and we compare them to Bot's reported results (Bot, 2001).

Experiments developed in (Bot, 2001) can be divided in three groups depending on the classifier used. In the first case, a Minimal Distance to Means classifier (MDM) was used. The second set of experiments employed a *k*-NN classifier with k=3. Finally, a classifier called parallelepiped was the element used in the third set of experiments. This last classifier was replaced by a calculation of fitness function based on Fisher criterion.

4.4.3.1 Results with *k*-NN classifier

Table 4.5 shows the results obtained from the implementation of Bot's method with Open BEAGLE as evolutionary tool, when fitness measure is based on *k*-NN classifier accuracy for each data set (first column). The second column shows the accuracy of the *k*-NN classifier alone applied to the original normalized data as reported in (Bot, 2001). This column is split in two parts: the recognition rate expressed in percentage with the standard deviation after the symbol "±" and the original number of features for each data set. For comparison purposes, the results reported in (Bot, 2001), transcript in the third column, are called "Bot's results for k-NN". In turn, it is also divided in the same two parts. The last column presents the results of our implementation of Bot's methods with Open BEAGLE. Results are shown in column named "results of OB implementation with k-NN", including the recognition rate and the standard deviation and average number of features. Preliminary tests were performed with different number of neighbours (k =1, 3 and 5) and k=3 showed the best trade-off between performance and time. The same value of neighbours was used in (Bot, 2001).

Table 4.5

Comparison Bot's results with the implementation in Open BEAGLE using *k*-NN classifier
Average Rec. Rate (%) ± standard deviation, using 5 repetitions and 10-fold cross-validation

Database	*k*-NN accuracy on normalized original data		Bot's results for *k*-NN		Results of implementation in OB with *k*-NN	
	Rec. Rate (%) ± Std. Dev	Orig. #F	Rec. Rate (%) ± Std. Dev	#F*ave*	Rec. Rate (%) ± Std. Dev	#F*ave*
Australian	81.2 ± 4	14	83.12 ± 6.89	1	74.35±24.68	1.4
Bupa	49.6 ± 7.2	6	58.4 ±11.53	1.5	67.94±29.97	1.7
Car	84.6 ± 4.5	5	79.1 ±8.36	1	74.26±12.36	1.7
Cmc	52 ±18.5	9	51.3 ±18.33	1.5	48.63±13.76	1.8
Ecoli	80.8 ±15	7	73.5 ±17.95	2.6	72.18±10.01	2.7
German	64 ±5.8	24	63 ±9.36	1.2	70.86±5.36	1.9
Glass	53.92 ±17.1	9	52 ±18.35	2.7	46.29±17.96	2.8
Ionosphere	74.92 ±9.6	34	80 ±6.48	1.3	76.57±19.96	1.4
Iris	92.72 ±6.6	4	94.1 ±7.88	1	94.66±7.25	1.8
Pima	70.22 ±4.3	8	69.5 ± 6.92	1.2	71.27±5.21	1.75
Segmentat.	89.72 ±2.8	19	88.4 ± 4.01	2.8	82.46±4.71	2.4
Sonar	71.82 ±19.6	60	72.2 ± 19.08	2.3	72.80±11.30	2.8
Teaching	40.52 ±24.6	5	42.5 ± 22.99	1.7	55.33±13.62	2.3
Waves	69.12 ±10.1	21	61.8 ± 14.87	2.9	66.06±16.58	2.7
Wine	88.32 ±10.9	13	87.6 ± 9.53	1.6	80.00±21.46	1.4
Yeast	48.82 ±5.9	8	41.9± 8.66	2.8	36.76±13.49	3.2

Results presented in Table 4.5 correspond to the average best individual with validation based on Bot's stopping criterion. The average is taken over all 10 folds and 5 repetitions. Results are consistent with the reported performances in (Bot, 2001).The average number of evolved features is greater in our implementation, which generates a slightly better

performance. In most of the cases, results are also comparable to the original recognition rate with a *k*-NN classifier when using the whole set of features. This is one of the key points of this method, because it permits to obtain similar, and in some cases better, recognition rates with a considerable reduction in the number of evolved features used. The GP-evolved features obtained with Bot's method establish a convenient transformation of the original data into a representation of a dimensionality equals to the required evolved features and a similar performance.

4.4.3.2 Results with MDM classifier

Table 4.6 compares results obtained by Bot and the implementation in Open BEAGLE when using a Minimal Distance to Means (MDM) classifier. Experiments have been run using 10-fold cross-validation with 5 repetitions. Average recognition rate, standard deviation and average number of evolved features are included. In our experiments we present the average performance of best individuals when validation is carried out at the end of the evolution. Most of the cases, the results are close each other in recognition rate and average number of features. In general, the results with Open BEAGLE give a slightly higher recognition rates, but at the same time, a higher average number of features. These values depend on the stopping threshold *d* which indicates where the evolution can be stopped because enough information from the evolved features has been retained to classify the data. With higher average number of features –in comparison to Bot`s results-, which means stopping later, greater amount of information is taken and then higher recognition rates can be obtained. Differences in performance are very small, they can be considered equivalent.

Table 4.6

Comparison Bot's results against our implementation using MDM classifier Average Rec. Rate (%) ± standard deviation, using 5 repetitions and 10-fold cross-validation

Database	Bot's results with MDM		Results of implementation in Open BEAGLE with MDM	
	Rec. Rate (%) ± Std. Dev	#F*ave*	Rec. Rate (%) ± Std. Dev	#F*ave*
Australian	82.5 ± 10.01	1	87.39 ± 3.06	1.3
Bupa	61.7 ± 11.51	1	63.24 ± 12.10	1.4
Car	73.1 ± 8.81	1.2	81.82 ± 4.75	1.7
Cmc	52.4 ± 18.08	1.6	52.18 ± 12.97	2.8
Ecoli	76.1 ± 14.77	2.6	72.42 ± 11.79	3.1
German	61.5 ± 12.98	1.1	66.76 ± 5.59	1.3
Glass	60.3 ± 15.25	2.1	58.10 ± 14.16	2.2
Ionosphere	83.1 ± 8.51	1.3	82.57 ± 6.42	1.1
Iris	95.3 ± 7.04	1	95.06 ± 5.84	1
Pima	68.9 ± 7.08	1	66.85 ± 5.48	1.07
Segmentation	91 ± 4.5	3.1	94.76 ± 4.16	1.8
Sonar	75.5 ± 18.83	1.3	66.50 ± 8.83	1.8
Teaching	51.8 ± 19.79	1.6	56.00 ± 10.36	1.7
Waves	68.1 ± 13.74	2.3	75.76 ± 7.6	2.1
Wine	88.1 ± 7.64	1.8	91.11 ± 5.97	1.8
Yeast	46.2 ± 4.89	3.9	50.00 ± 8.92	4.4

4.4.3.3 Results with fitness based on Fisher criterion

The method implemented follows the principles of Bot's algorithm with a fitness measure based on Fisher criterion. Table 4.7 presents the original accuracy of a k-NN classifier, the results reported in (Bot, 2001) and our results. Each of these three parts is divided in two columns: recognition rate ± standard deviation and average number of evolved features. This fitness measure tries to maximize the inter-class scatter over the intra-class scatter using the Fischer criterion. This is done with small but continuous increases along with generations as will be shown above. As shown in equation 4.6 there are two parameters to adjust when using this fitness measure. Parameter λ is a factor that considers the contribution from the mean value. For instance, if the minimum values for two features are similar, the one with the largest average of values will survive. It was empirically set to 0.001 as indicated in (Guo *et al.,* 2005) because there were not considerable changes with different values of λ. In the other hand, the scaling factor β was introduced to maintain a level of increase of fitness measure. For instance, in data sets with poor performance the fitness values can be very small with even smaller changes along with generations that result in minimum and sometimes negligible increase in recognition rates. With greater fit_{Fish} values (see equation 4.6), progress is faster with better results at the end. In the same way, when classification of some data sets is not that difficult, the parameter β helps to avoid that evolution gets trapped very soon in a local maximum. In these cases, β is set to values around 20 to 30.

The fitness measure based on Fisher criterion was applied in (Guo *et al.,* 2005) for a large number of generations: 1000 (compared to the number used in our case).

Recognition rates are very similar to values reported in (Bot, 2001) and our results when the fitness measure is based on a *k*-NN. Standard deviation values are smaller (see Table 4.7), suggesting that the inter-class scatter over the intra-class scatter increase is made and the method is stable over different data set partitions. As in previous cases, the average number of evolved features is somewhat higher than values reported in (Bot, 2001). In any case differences are not considerable.

Table 4.7

Average Rec. Rate (%) ± standard deviation of Bot's algorithm with fitness based on Fisher criterion and a *k*-NN, using 5 repetitions and 10-fold cross-validation

Database	*k*-NN accuracy on normalized original data		Bot's results for *k*-NN		Bot's method with Fisher fitness and *k*-NN classifier	
	Rec. Rate (%) ± Std. Dev	Orig. #F	Rec. Rate (%) ± Std. Dev	#F*ave*	Rec. Rate (%) ± Std. Dev	#F*ave*
Australian	81.2±4	14	83.1 ± 6.89	1	88.70 ± 2.04	1.5
Bupa	49.6±7.2	6	58.4 ± 11.53	1.5	61.46 ± 1.29	1.4
Car	84.6±4.5	5	79.1 ± 8.36	1	77.91 ± 16.26	1.8
Cmc	52.0±18.5	9	51.3 ± 18.33	1.5	55.94 ± 9.54	2.3
Ecoli	80.8±15	7	73.5 ± 17.95	2.6	77.91 ± 8.65	2.1
German	64.0±5.8	24	63 ± 9.36	1.2	68.12 ± 3.20	1.9
Glass	53.9±17.1	9	52 ± 18.35	2.7	58.57 ± 12.24	2.5
Ionosphere	74.9±9.6	34	80 ± 6.48	1.3	82.60 ± 2.60	1.4
Iris	92.7±6.6	4	94.1 ± 7.88	1	93.45 ± 5.53	1.3
Pima	70.2±4.3	8	69.5 ± 6.92	1.2	72.84 ± 4.56	1.7
Segmentat.	89.7±2.8	19	88.4 ± 4.01	2.8	84.76 ± 7.71	2.5
Sonar	71.8±19.6	60	72.2 ± 19.08	2.3	76.50 ± 7.83	2.4
Teaching	40.5±24.6	5	42.5 ± 22.99	1.7	55.68 ± 5.79	1.3
Waves	69.1±10.1	21	61.8 ± 14.87	2.9	67.20 ± 4.56	2.1
Wine	88.3±10.9	13	87.6 ± 9.53	1.6	90.69 ± 3.28	1.7
Yeast	48.8±5.9	8	41.9 ± 8.66	2.8	55.95 ± 4.82	2.7

Table 4.8

Bot's algorithm using fitness based on Fisher criterion and a MDM classifier Average Rec. Rate (%) ± standard deviation, using 5 repetitions and 10-fold cross-validation

Database	Bot's results with MDM		Bot's method with Fisher fitness and MDM Classifier	
	Rec. Rate (%) ± Std. Dev	#F*ave*	Rec. Rate (%) ± Std. Dev	#F*ave*
Australian	82.5±10.01	1	87.83±15.57	1.1
Bupa	61.7±11.51	1	65.59±12.78	1.7
Car	73.1±8.81	1.2	76.36±12.54	1.5
Cmc	52.4±18.08	1.6	46.33±3.67	2.1
Ecoli	76.1±14.77	2.6	61.47±11.38	1.6
German	61.5±12.98	1.1	71.50±4.83	2
Glass	60.3±15.25	2.1	45.24±11.05	1.5
Ionosphere	83.1±8.51	1.3	81.14±5.59	1.5
Iris	95.3±7.04	1	96.00±6.32	1.1
Pima	68.9±7.08	1	68.24±3.30	1.3
Segmentation	91±4.5	3.1	86.67±7.7	1.9
Sonar	75.5±18.83	1.3	62.50±9.50	1.1
Teaching	51.8±19.79	1.6	53.33±10.42	1.3
Waves	68.1±13.74	2.3	74.24±10.02	2.7
Wine	88.1±7.64	1.8	89.44±4.86	1.1
Yeast	46.2±4.89	3.9	39.19±8.03	3.1

Table 4.8 presents the results reported in (Bot, 2001) and our results when fitness measure is based on Fisher criterion and final performance is calculated with a MDM classifier. Results obtained when are very close to outcomes from Bot method with MDM classifier performance as fitness measure. Average number of features is similar as well. Standard deviation is higher with Fisher criterion as fitness measure is also higher.

Computation time with Fisher criterion as fitness measure is much faster than with a *k*-NN classifier but a slower than MDM classifier (as fitness measure). Calculation of fitness measure based on Fisher criterion requires calculation of mean and variance for each proposed individual and a final classification. In Bot's method only mean is calculated for each proposed individual and classification is done with MDM classifier as well. In both cases computation is linear with the number of samples and faster than *k*-NN classifier (Bot, 2001). We conclude that fitness measure based on maximization of inter-class scatter over the intra-class scatter is an adequate measure for the UCI data sets used. The maximization of the inter-class scatter over the intra-class scatter using the Fisher criterion is done with small but continuous increases as the evolution progresses.

Analysis of Fisher variation with generations and features

During training phase, the system apprehends with the evolution, i.e. after each generation the system gains knowledge about the problem. Fitness in terms of Fisher criterion measures the distribution of the interclass scatter over the intra-class scatter for any two classes (Guo *et al.,* 2005). In Bot's method implementation with Fisher function as fitness measure for Australian dataset, fitness has a slight increment as evolution goes through new generations, as shown in Figure 4.4. This measure generates a recognition rate that is calculated at the end of the evolution cycle. Figure 4.4 illustrates how the fitness values change with the number of generations and with the number of features added.

In the other hand, there is a clear increment of Fisher fitness with the number of features. Incremental gain in fitness is lower as the number of features increases. This can be seen through the projection of surface curves in the plane Generations-Features in Figure 4.4. The separation between lines (surface projection as lines in plane Generations-Features) increases with the number of features. In other words, the slope of the curve is greater for evolved features 1, 2, 3 and diminishes as the number of features increases –the slope or incremental fitness decreases-. As the number of generations increase, the recognition rate also increases but the increment is very small.

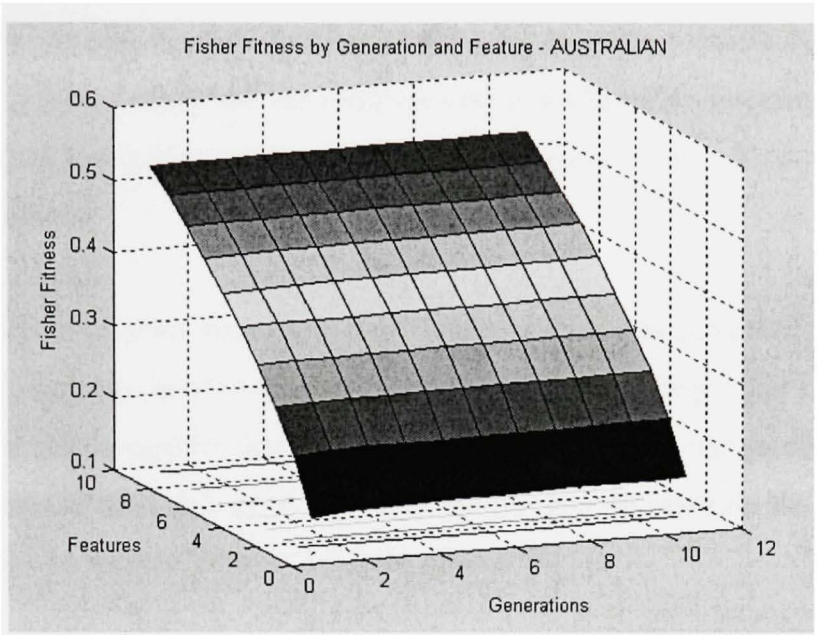


Figure 4.4 ***Fisher fitness values by generations and evolved features. Australian data set.***

4.4.3.4 Performance for unbalanced data sets

Most of the data sets with an unbalanced number of samples between classes, as in the case of Ecoli, produce a high standard deviation. This unbalance causes very different recognition rates depending if the less populated classes are included or not in some training folds. The worst case scenario, as mentioned before, are classes with just a few samples that can be part of optimization, validation or even test folds and not included at all in the corresponding training fold. Recognition rate in these cases is decreased in consequence. On the other hand, when the few samples are part of the training fold and no any sample appears in validation, optimization or test folds, the classifier performance is also affected.

4.4.3.5 Representation of data sets with one or two evolved features

One of the advantages of Bot's method is to construct a small set of evolved features that are able to represent a data set. For instance, Ionosphere data set is originally composed of 34 raw features. After construction of evolved features, the data set can be represented with an average of 1.1 features. Figure 4.5 maps the values of the data set for two evolved features

EF(1) and EF(2). As the average number of features is greater than 1.0, we display two features as if they were orthogonal (or independent) in a Cartesian diagram. The aim of this figure is to analyze the quality of the results obtained and permit a simple visualization of Bot's method potential.

As can be seen in the graphic, data from class 0 (blue **x**'s) is well separated from class 1 data (red +'s). In fact, variance in the horizontal axe (EF(1)) is much larger that in the vertical axe (EF(2)). Such variance suggests that Ionosphere dataset could be represented with only one feature. In addition, Figure 4.5 gives a hint about the way to separate the two classes, for instance with the two vertical lines (only in the axe of EF(1)).

Figure 4.5 shows a detail of the mapping in two features. The interval 15 < EF(1) <= 30 is analyzed. The majority of the samples that are overlapped are comprised within this interval. The two suggested vertical lines (axe of evolved feature 1) show that classes can be very well separated and, in fact, only one feature could be used with very good results. Such a way of mapping evolved features reflects the efficacy of Bot's method to separate classes. A complete classification requires to determine value of raw features (terminal set elements) that generate 15 < EF(1) <= 30.

The first evolved features EF(1) is displayed in Figure 4.6. It is an individual with a size of 63 nodes, and a depth of 16 that achieves a fitness of 92.06% in the optimization data set. The second feature is very simple, a division of raw feature 29 over raw feature 8: IN29/IN8 that is a tree size of 3 and a depth of 2, for a fitness of 92.38% (this evolved feature is not shown). A quick analysis of EF(1) in Figure 4.6 shows that the sub-tree is (IN4 + IN0) is repeated through the tree at different depths of EF(1), so it can be considered as a building block of the solution . There are variants of this tree: addition of raw feature IN5 or other sub-tree to the head of the main building block.

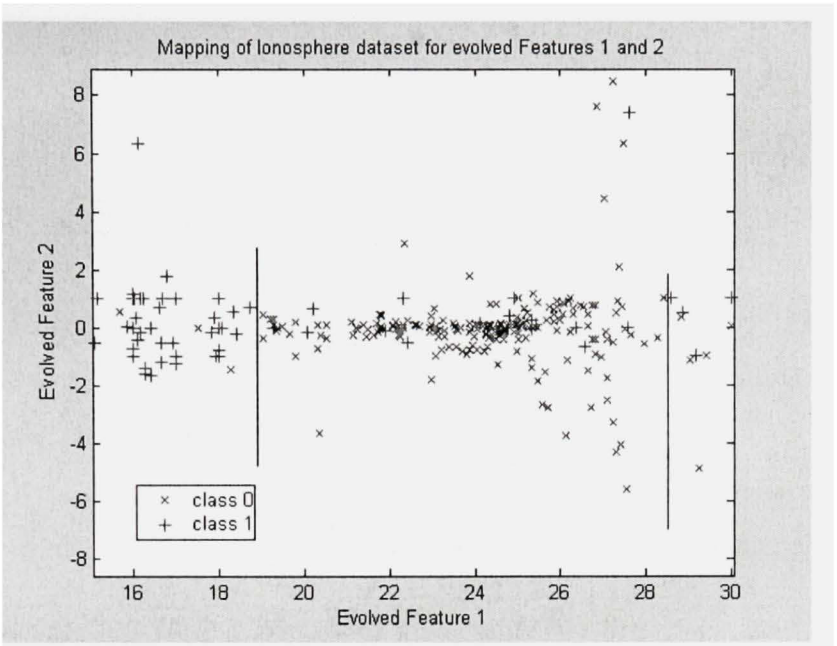


Figure 4.5 ***Analysis of representation of Ionosphere data with one or two evolved features***

Looking the evolved feature 1, we realize that only three raw features are involved in the expression: IN0, IN4 and IN5. And as mentioned, EF(2) is composed of IN29 and IN8. Ionosphere data set is composed of 34 raw features IN0, IN1,..., IN33 and Bot's algorithm generates only two evolved features with a classification rate of 92.38%. That is a huge reduction in the representation with good performance results. Another point to mention is the number of raw features used in the whole solution. In the example presented here, only five raw features are used to produce the two evolved features from a starting space of 34 features (see Table 4.1). Therefore we see that the feature construction method involves, in this case, a selection of the most important features at the same time. A more complete analysis about the raw features used to build the evolved features is presented in chapter 5.

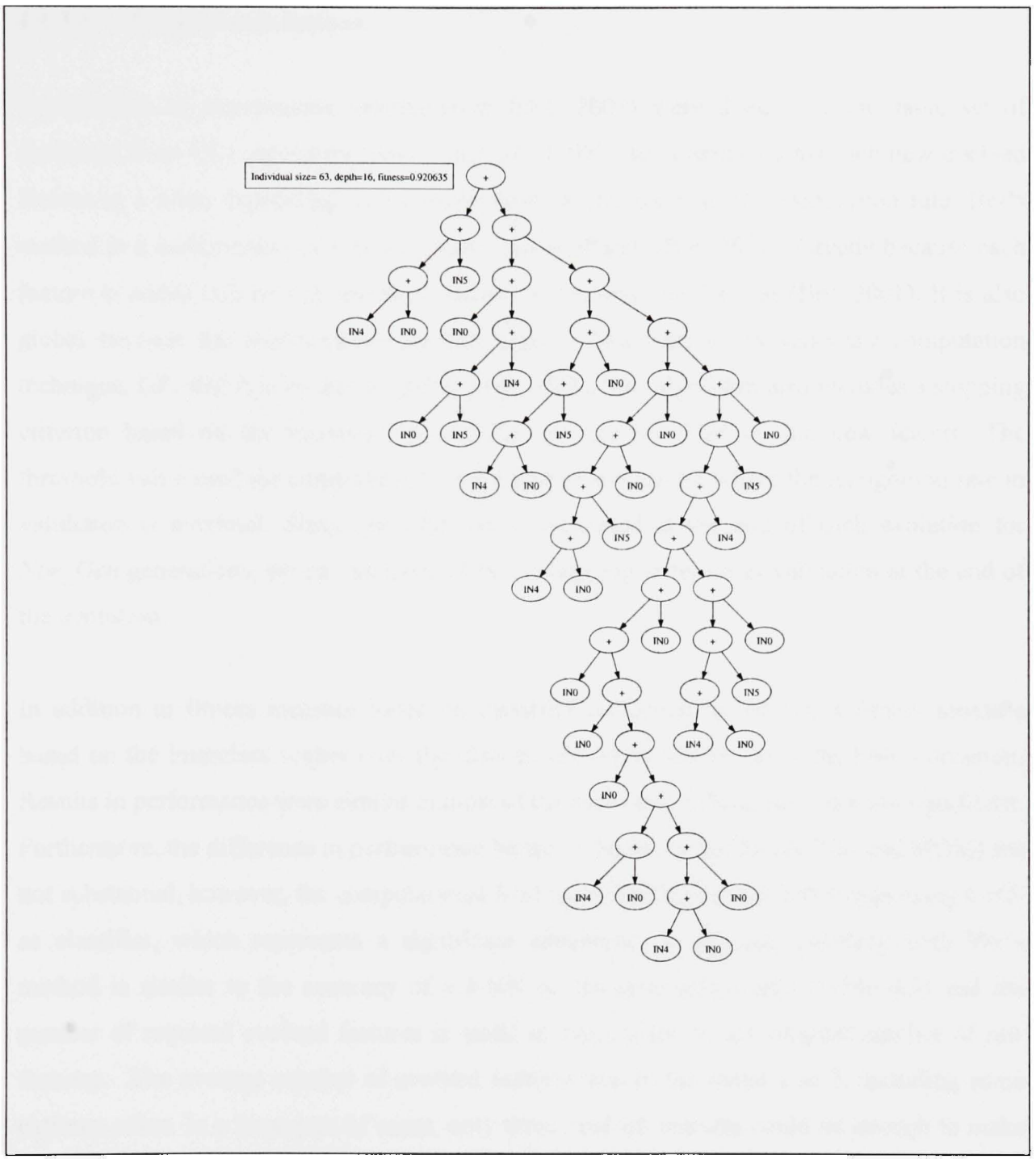


Figure 4.6 ***Evolved feature 1 for Ionosphere example***

4.4.3.6 General conclusions

Replications of experiments developed in (Bot, 2001) were done over the same set of databases from UCI repository (Newman *et al.,* 1998). Bot's method adds each new evolved feature at a time, depending on its incremental contribution to the recognition rate. Bot's method is a combination of a greedy and a global search (Bot, 2001). Greedy because each feature is added one by one and each feature is the best found so far (Bot, 2001). It is also global because the searching mechanism used is based on an evolutionary computation technique, GP, which is by nature global (Bot, 2001). The algorithm also includes a stopping criterion based on the increment in recognition rate provided by the new feature. The threshold value used for comparison is calculated at the feature where the recognition rate in validation is maximal. Since this criterion is evaluated at the end of each evolution for *Nbr_Gen* generations, we can understand Bot's stopping criterion as validation at the end of the evolution.

In addition to fitness measure based on classifier performance, we use a fitness measure based on the interclass scatter over the classes intra-class scatter using the Fisher criterion. Results in performance were similar in most of the cases and differences were not significant. Furthermore, the difference in performance between the two classifiers (k-NN and MDM) are not substantial, however, the computational load using MDM is much lower than using k-NN as classifier, which represents a significant advantage. In general, accuracy with Bot's method is similar to the accuracy of a k-NN on the normalized data (Table 4.5) and the number of required evolved features is small in comparison to the original number of raw features. The average number of evolved features was in the range 1 to 7, including some extreme cases. In a large part of cases, only three evolved features could be enough to make an adequate classification. In addition, the number of raw features used in the evolved features is small. That means that the feature creation process involves, at the same time, a feature selection mechanism.

CHAPTER 5

IMPROVEMENTS TO FEATURE CREATION FOR CLASSIFIERS

In this chapter we propose three different procedures to improve Bot's performance. The GP algorithm run in Chapter 4 used the same parameter setting as in (Bot, 2001), to make a fair comparison of results. We analyze if settings for parameters such as: population size, number of generations and probabilities of genetic operations cross-over and mutation, were defined in suitable ranges. We do not try here to optimize GP parameters (for example those presented in Table 4.2) because this is a monumental and very time consuming task. Instead we try to find ranges of values where the algorithm has an acceptable performance. In Section 5.1 we explain these parameters and we evaluate their influence with by means of some experiments run for different combinations of cross-over and mutation probabilities.

The second procedure developed in this chapter is based on the observation that running the experiments in Chapter 4 is very consuming in time and resources, even for small data sets as those used until now. We then considered some alternatives to speed up the computation of solutions maintaining the quality of results. One way is to parallelize the GP algorithm. We present in Section 5.2 different alternatives, analyze them with regards to our algorithm and decided by a method called coarse-grain or island. Once more, we tested the method with some data sets employed in Chapter 4. These test are explained, their results presented and analyzed in Section 5.2.3

The third procedure is focused on the generalization power of solutions proposed. Bot's method adds each new evolved feature at a time, depending on its incremental contribution to the recognition rate. A new evolved feature is accepted if the new relative gain is greater than a threshold. The threshold value used for comparison is calculated at the feature where the recognition rate in validation is maximal. This is the stopping criterion applied in Bot's algorithm. Since this criterion is evaluated at the end of each evolution for *Nbr_Gen* generations, we can understand Bot's stopping criterion as validation at the end of the

evolution. We have to note anyway that the methods are not exactly equivalent because the verification is done for every evolved feature and not at the end of the whole evolutionary process. Also, if at any evolved feature the relative increment in recognition rate is not greater than the threshold value, this last evolved feature is discarded. As a result, it is less likely that the final solution has going too much into over-fitting because the last feature was completely discarded. We said that the prompt discard of the feature reduces the chances of accepting over-fitting solutions.

Even though Bot's stopping criterion tries to reduce the over-fitting towards the optimization data set, validation is only made at the end of each evolving feature and not in between the features. In Section 5.3 of this chapter, we present a method called global validation that allows us to identify if there is over-fitting by evaluating the generalization power of each individual in the population for every feature and generation by mapping them into the validation data set and keeping trace of the best individual there (in validation). We present, different types of validation and explain in detail the global validation procedure, initially proposed in (Radtke *et al.,* 2006). In addition we explain how to apply it to Bot's method. We tested the global validation procedure in some of the data sets employed in Chapter 4. These test are explained, their results presented and analyzed in Section 5.3.6.

As mentioned before, Bot's method creates evolved features based on GP and at the same selects the raw features that are more important in classification. We present an analysis of the selection of raw features through the evolution and at the final solutions.

Experiments were carried out on a Beowulf cluster with 25 nodes using Athlon XP 2500+ processors with 1GB of PC-2700 DDR RAM. Finally, Section 5.5 presents some general conclusions of the optimization applied to Bot' method for creating evolving features.

5.1 Analysis of some GP parameters

In Chapter 4, the experiments were run using the parameter settings as indicated in (Bot, 2001), to make a fair comparison of results. The individual probability cross-over was set to 0.9 and the individual probability mutation was set to 0.1 (Bot, 2001). The analysis of the best combination of these two probabilities is by itself a difficult task. Koza used the same set of parameters for the vast majority of the problems analyzed in (Koza, 1992; Koza, 1994; Koza *et al.*, 2005). Thus allow him to give the benefits of the results to the GP and, as he mentions, not to an intricate tailoring of these parameters (Koza *et al.*, 2005). Furthermore, Banzhaf *et al.* point out that GP works over a wide range of parameter settings and instead of doing an exhaustive analysis of the best settings they give recommendations about typical values that have been applied to different cases with positive results (Banzhaf *et al.*, 1998). Fernandez *et al.* state that tuning of the parameters has to be done one at a time, even though that gives sub optimal results, because parameters are interdependent and interact in complex ways (Fernandez et al., 2003). They investigate suitable parameter ranges, rather than optimal values (Fernandez *et al.*, 2003). In this section we follow these suggestions.

5.1.1 Population size and number of generations

Population size was set to 100 individuals and we used 11 generations for each feature (Bot, 2001). It is known that bigger populations have more genetic diversity, therefore explore more areas of the search space which may lead to find out the desired solution with fewer evaluations (Banzhaf *et al.*, 1998). Also, bigger populations take more time to evolve (Banzhaf *et al.*, 1998). Populations starting at 100 individuals for "small" problems and 10000 individuals for difficult problems are suggested in (Banzhaf *et al.*, 1998). From his side, Koza used populations of 500 individuals in "small" problems and increase it to 500,000 to very complex problems (Koza et al., 2005). Anyway, we have to remember that he can have populations of these sizes because he uses a 1,000-nodes cluster to run the experiments. We use a population of 100 individuals as indicated in (Bot, 2001) and the results were very close to those reported in (Bot, 2001). Bigger populations could possibly generate better results but the computation time will be increased in such a way that the

applicability of the method to greater data sets becomes compromised. The number of generations was fixed to 11. Different numbers of generations, from 11 to 30, were tested and we did not see big differences in the results. The small progress reached with more generations is easily attained with the addition of a new feature. The number of generations used in Chapter 4, again as indicated in (Bot, 2001) is one of the main advantages of Bot's algorithm. This algorithm prevents from long runs and still gives results similar to those obtained with long runs. Consequently, we will use 11 generations per feature (thus takes into account the generation 0 randomly generated)

In order to make it clear the reference to the diversity in the population in GP-based evolutions we present the definition used and the way to measure it. The measures of diversity are concerned with the levels and types of variety on populations (Burke *et al.,* 2004). They can be defined over fitness values (phenotypic), individuals (trees) structures (genotypic) or a combination of both (Burke *et al.,* 2004). Since the measure of success in evolutionary algorithms is the fitness of a solution in the problem's environment (Burke *et al.,* 2004), we take it to define the diversity measure used here. Rosca defined the entropy of a population as a function of fitness values in it. Entropy represents the amount of disorder in the population (Burke *et al.,* 2004; Rosca, 1995), so population entropy corresponds to diversity within the population (Rosca, 1995). Phenotypic diversity is related to the number of different fitness values in the population, it is noted as H(P) and defined as:

$$H(P) = -\sum_{j=1}^{\eta} fit_j \log(fit_j), \tag{5.1}$$

where fit_j is the fraction of individuals in P having a fitness j and η is the number of different fitness values in P. In general, low entropy means low diversity in the population, but the values can be interpreted in terms of the number of different groups having the same fitness values (Folino *et al.,* 2004). Thus, high entropy can be considered as a high number of small groups, each one having the same fitness value whereas low entropy would be considered as a large number of groups having the same fitness (Folino *et al.,* 2004). In the experimental

part we will show how the entropy varies along the evolution. Entropy graphs will show the diversity of the population when at each new evolved feature.

5.1.2 Fitness function

In Chapter 4, we worked with two different types of fitness function. The first type was the recognition rate of a classifier. We used *k*-NN and MDM classifiers. Recognition rate is measured by applying (or interpreting) the individual selected as solution to the data set and the new set of values is the input to the classifier which calculates the recognition rate level. During the evolution, the individuals selected as solution are applied to the optimization data set and during test, individuals is interpreted in the test data set. The interpretation of the individuals with a data set, transforms it into a vector of values, one for each sample. The size of the vector values for each sample depends on the number of evolved features.

We see that during the evolution each of the individuals has to be interpreted to calculate its fitness. The same occurs during the global validation procedure. A general behaviour of a GP algorithm, is that individuals grow in size (number of nodes) and depth (number of levels of the tree), along with the generations. So, the interpretation adds more and more time to the fitness computation. This increment in the size of individuals falls into what is called *code bloat*. That is, individuals grow uncontrollably until the maxima allowed depth and size. These individuals, in general, are characterized by having large portions on them that are in close proximity of high fitness blocks of code (Banzhaf et al., 1998). Their big size and large depth make them less vulnerable against genetic operations. In consequence, they obstruct the evolution and a possible result is the stagnation of the evolution. Normally, these individuals are compose of GP *introns* (Banzhaf et al., 1998), which is code that is not useful (for example a+0 or a*1 or a/1) but makes them more resistant against changes during the evolution. During the analysis of our runs we have detected this problem and we used a mechanism to diminish the effect of code growth. The mechanism is based on adding a penalty within the fitness function based on the size of the individual. For a fitness function based on recognition rate, the penalty is proportional to the inverse of the size of the

individual (number of nodes) and is subtracted from the recognition rate calculated by the classifier (see equation 5.2). To fix the proportional constant γ, we tested different values in the range 10 to 100 and found the most appropriate value 100. This value has to be adjusted as to avoid the uncontrollable growth without affecting too much the original result of the classifier. This modification effectively controlled the growth in individuals because bigger individuals with the same blocks of high fitness were penalized. It pushes the evolution towards better and smaller solutions.

$$fit_{pena} = fit - \frac{Tree\_size}{\gamma}, \qquad (5.2)$$
$$\gamma_{Best} = 100,$$

A second type of fitness function maximizes the inter-class scatter over the intra-class scatter using the Fischer criterion (Duda *et al.*, 2001; Guo *et al.*, 2005). The fitness function measure used was specified in equation 4.6. A constant λ empirically set to 0.001 in (Guo *et al.*, 2005) is a factor that considers the contribution from the mean value. For instance, if the minimum values for two features are similar, the one with the largest average of values will survive. In addition, a value β is experimentally applied to re-scale the resulting fit_{Fish} values less than a maximum fitness value set during evolution (100 in this case). In addition, we have to add the penalization factor to avoid an uncontrollable growth of individuals. Equation 5.3 presents the equation 4.6 after applying the penalization factor. Each of the constants are explained above.

$$fit_{Fish-pena} = \frac{1}{\beta}\left(\min(fit) + \lambda.\frac{1}{q}\sum_{i=1}^{q}(fit)\right) - \frac{Tree\_size}{\gamma}, \qquad (5.3)$$

Initially we tried different values for constants λ and β and then we add the penalization factor setting appropriately the γ constant. Value of β scale is adjusted according to each data set. Different values for constants λ and β were tested. Values in the range 0.0001 to 0.01 for constant λ and values in the range 5 to 50 for β were tested for the data sets of Chapter 4. After many experiments we see that setting a good set of values for constants λ and β at the

same time is a difficult task, when Fisher based fitness is applied to Bot's method. As mentioned in Chapter 4, the changes in Fisher based fitness values are very small along with the increase in generations. Since Bot's method employs just 11 generations, the difference were very small from the beginning to end of the evolution of a feature (in contrast Guo and Nandi run evolutions for 1000 or 10000 generations). To change that and make stronger the changes in fitness values, we have to increase the β constant. Further analysis showed that β constant diminished the influence of λ constant and it is not easy to find a very good combination of these two constants, so we let λ in its default value 0.001 and show the best set of β values in Table 5.1

Table 5.1

Values for constant β for fitness function with Fisher criterion with constant λ=0.001

Value constant β in equation 5.2	**Data sets**
5	Cmc, Yeast, Glass, Teaching
10	German, Car, Ecoli, Sonar, Ionosphere, Pima, Australian, Waves, Bupa, Wine
20	Iris, Segmentation

When the fitness in Bot's method was based on Fisher criterion, individuals proposed had a huge size in comparison to individuals generated when fitness was measured with the recognition rate of a classifier (*k*-NN or MDM). Therefore, the penalization due to individuals' size (tree_size) is very important here. The best value of γ (=100) found before was not longer applicable in this case. We tested different values and found that γ =1000 was more appropriate for equation 5.3.

Setting of fitness function based on classifier accuracy is simpler and straight forward than using the Fisher criterion, because it does not need the adjustment of different constants that depends on each data set used. Further analysis of equation 4.3 reflects that a large value of *fit* may be due to well separated clusters or to two overlapping classes with small variances

(Guo and Nandi, 2006). The case of overlapping increases the classification error. Guo and Nandi present in (Guo and Nandi, 2006), an alternative fitness measure in which the within-class scatter uses a distance between any two patterns of the same class instead of the variance. As a result, for subsequent experiments we only use fitness function based on the classification error (k-NN or MDM) and not fitness based on the Fisher criterion.

5.1.3 Cross-over and mutation probabilities

Table 4.4 indicates that individual cross-over probability (*gp.cx.indpb* parameter) was set to 0.9 and individual mutation probability (*gp.mutstd.indpb* parameter) was set to 0.1. The analysis of the best combination of these two probabilities is by itself a difficult task. (Banzhaf *et al.*, 1998) indicate as a rule of thumb start with 0.9 cross-over probability and 0.1 mutation probability. Table 4.4 gives the definition of different types of cross-over and mutation that takes part during the evolution. We present here some examples of parameter settings to show their meaning and interdependence. For instance individual cross-over probability of 0.9 means for a population of 100 individuals, that 90 individuals (40 pairs) from each generation are selected (with reselection allowed) to participate in cross-over (Koza, 1992). The parameter cross-over distribution probability (*gp.cx.distrpb*) refers to the selection of the point (within the tree) to make the cross-over. It indicates the probability that a crossover point is a branch (node with sub-trees). A value 1.0 means that all crossover points are branches, and a value 0.0 means that all crossover points are leaves (Gagné and Parizeau, 2004a). The setting in 0.9 promotes the recombination of larger structures instead of mere swapping of terminals (probability values closer to 0.0) (Koza, 1992).

Probabilities related to mutation are more varied. In the standard mutation (individual) a sub-tree is replaced with a randomly generated one. The point to do the mutation is selected with the aid of the parameter *gp.mutswap.distrpb*, which refers to the probability that a swap mutation point is a branch (node with sub-trees). A value 1.0 means that all swap mutation points are branches, meanwhile a value 0.0 means that all swap mutation points are leaves (Gagné and Parizeau, 2004a). This is set to 0.5 to have equal probabilities of mutation point

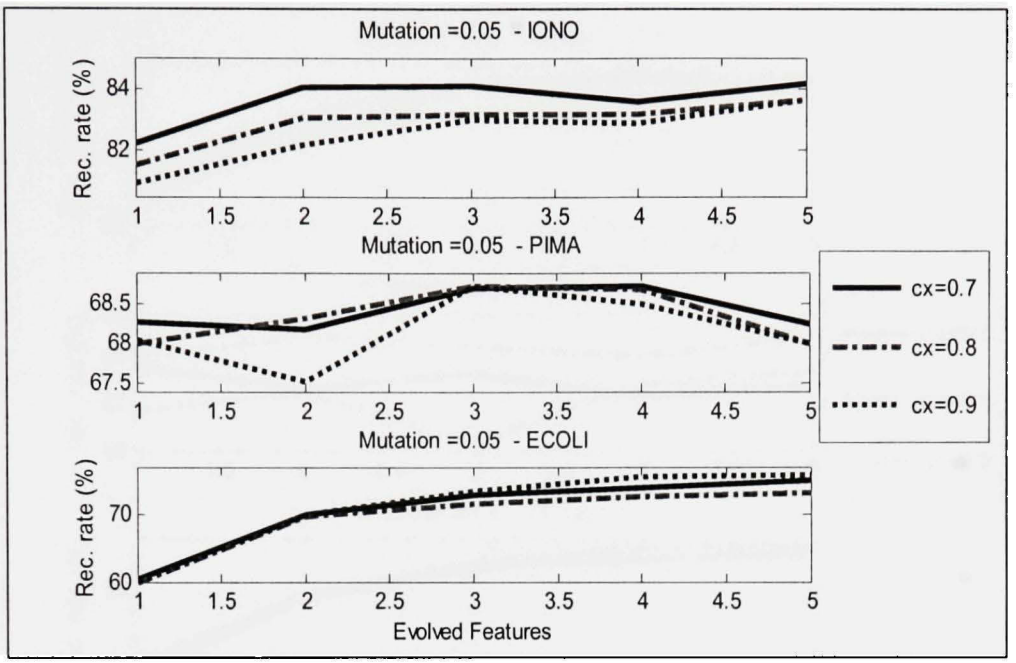
in branches than leaves. Two other parameters are related: the shrink mutation probability for an individual (*gp.mutshrink.indpb*) that consists in replacing a branch (a node with one or more arguments) with one of his child node. This erases the chosen node and the other child nodes (Gagné and Parizeau, 2004a) and swap mutation probability for an individual (*gp.mutswap.indpb*) that consists in exchanging the primitive associated to a node by one having the same number of arguments (Gagné and Parizeau, 2004a). All these additional parameters related to mutation operator have been set to its default value (see Table 4.4).

As shown, searching the optimal values for all these parameters becomes an optimization problem by itself, so we will try to find suitable ranges for individual cross-over and mutation probabilities.

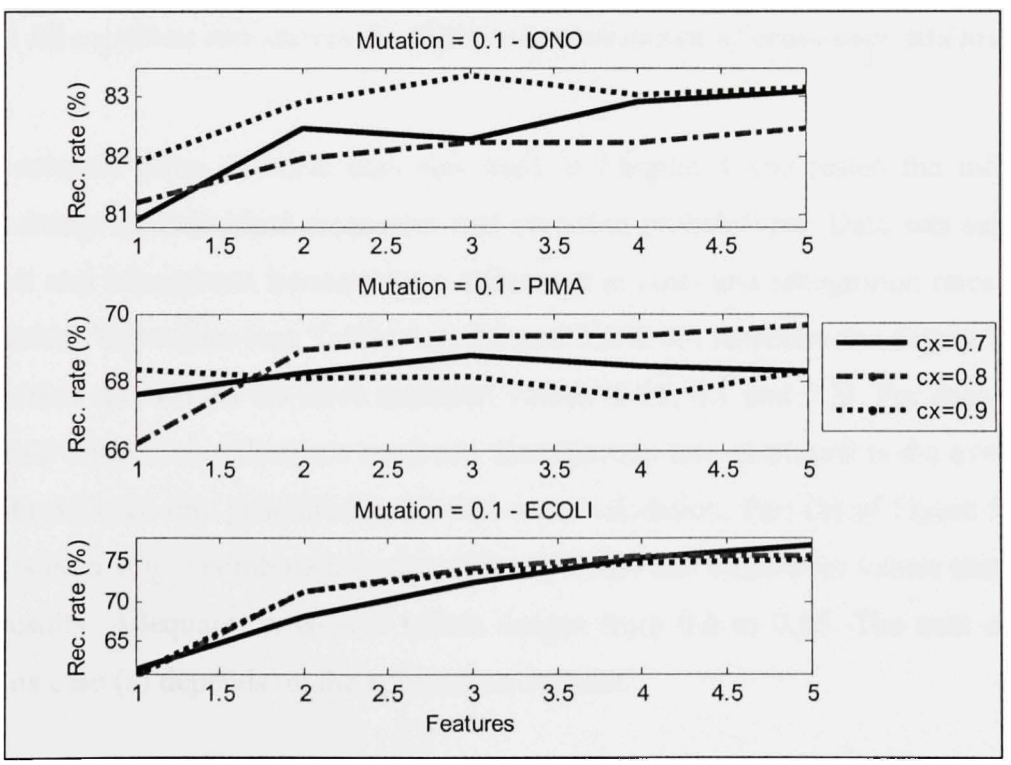
It is worth to mention that Michalewicz and Schmidt (Michalewicz and Schmidt, 2007) recently presented a new way to automatically and dynamically set these probabilities along with the evolution. The mechanism counts the number of times that an operator produces an off-spring that is better than all its parents and the success ratio for each operator can be calculated as the number of improved off-springs divided over all improved off-springs. The probabilities are set as the addition of the previous probability plus the success ratio multiplied by a constant in the range of zero to one. Initial probabilities are set to small values and they will be adjusted as evolution goes. If no improved off-springs are produced, the success ratio is not defined and probabilities are not adjusted (Michalewicz and Schmidt, 2007).

5.1.4 Experiments with cross-over and mutation probabilities

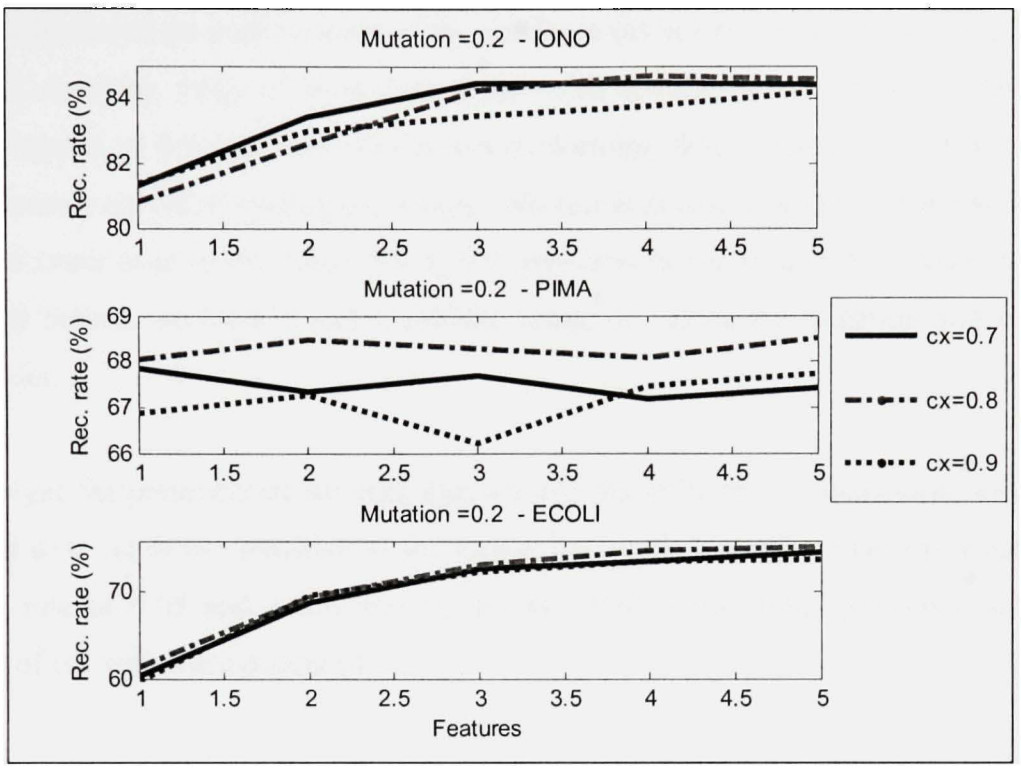
As mentioned before we will deal here with the individual cross-over and mutation probabilities and all other genetic parameters are maintained in their default values used in Open BEAGLE which agrees with the rules of thumb and suggestions in the literature (Banzhaf *et al.*, 1998; Koza, 1992; Koza, 1994; Koza et al., 2005).



(a) Mutation of 0.05 and different crossover values



(b) Mutation of 0.1 and different crossover values



(c) Mutation of 0.2 and different crossover values

Figure 5.1 ***Recognition rate curves for different combination of cross-over and mutation.***

We have selected three different data sets used in Chapter 4 and tested the influence of different settings of individual cross-over and mutation probabilities. Data sets selected are Pima, Ecoli and Ionosphere, because their difference in sizes and recognition rates achieved with the default GP values (see Tables 4.1, 4.5 and Table 4.4 respectively).Figure 5.1 shows the recognition rate values for three mutation values (0.05, 0.1 and 0.2). For each mutation setting, some cross-over values are analyzed. Recognition rate displayed is the average over 10 replications, each one generated as 10-fold cross-validation. Part (a) of Figure 5.1 shows that there is no a single combination of mutation (=0.05) and cross-over values that generate the best results. Adequate cross-over values ranges from 0.6 to 0.85. The best cross-over value in this case (a) depends on the application data set.

Figure 5.1(b) shows an improvement of the results in (a) but no single cross-over probability is the best. Anyway, range of cross-over is increased toward the interval 0.8 to 0.9. Finally, mutation equals to 0.2 produces results less performing than a mutation of 0.1, without a specific cross-over value leading the results. We can conclude that the best mutation setting is 0.1 and cross-over in the range 0.8 to 0.9 generates better results than other ranges. As mentioned before, we have found a suitable range of values for mutation and cross-over probabilities.

If we analyze the performance for each data set over the different combinations, we conclude that Ecoli data set is not sensitive to the ranges analyzed. Ionosphere performs better for a mutation rate of 0.05 and a low cross-over rate (0.6) whilst Pima performs better for a mutation of 0.1 with a cross-over of 0.8.

Since there is no a unique combination of mutation and cross-over probabilities that generates the best results, we set the mutation probability to 0.08 (in between the two best values) and the individual cross-over probability to 0.9 for further experiments, including new data sets to be used.

5.2 Parallelization of optimization

One of the advantages that offer GA and GP over classical optimization techniques is that they provide a set of solutions at a time, instead of a single solution. These solutions correspond to the population at each generation during the evolutionary process. A quick look to the general GA presented in Section 2.2 makes clear that they are in fact parallel algorithms. The same occurs with the GP algorithm presented in Section 2.3. In their initial times, they were implemented in sequential computers due to the lack of availability of parallel systems. Parallelization techniques in GA and GP are divided in three main groups: *master-slave*, *coarse-grained* and *fine-grained*. There are also hybrid techniques that we will not discuss in this document.

In master-slave parallelism, a master processor stores the whole populations and applies genetic operations (Cantú-Paz, and Goldberg, 2001; Gagne *et al.,* 2003). At each generation, the master processor distributes individuals to the slave processors for fitness evaluation. Thus, the speed up in fitness evaluation depends on the number of slaves (Gagne *et al.,* 2003). Therefore, parallelization of GA (or GP) using a *master-slave* mechanism will offer a considerable reduction in the total time required to run an evolution. This type of parallelism is applicable when the fitness evaluation can be done in parallel, such as in the generational replacement (see Section 2.2.3.4). Our implementation uses a steady-state replacement strategy, so a different parallelization alternative is necessary.

In the fine-grained model the fitness evaluation of individuals is distributed across a number of nodes (Vincent, 2003) that form a neighbourhood and it is calculated simultaneously for all of them (Folino *et al,* 2003). Nodes are interconnected in a two dimensional mesh or in a toroid. Selection, reproduction and mating are decentralized (Vincent, 2003) and take place within the neighbourhood. Information slowly diffuses across the grid, which generates semi-isolated niches of individuals having similar characteristics (Folino *et al,* 2003).

Coarse-grained model also called *island* method or *multiple-deme* (Cantú-Paz, and Goldberg, 2001) consists in evolving isolated sub-populations, named demes that occasionally exchange individuals in a migration process (Lin et al., 1994; Alba and Troya, 1999; Fernández *et al.*, 2003; Gagne *et al.,* 2003). Coarse-grained method can be applied to steady-state replacement strategies. Hybrid models use a coarse-grained approach at the top level and each deme is implemented in any other form of parallel algorithm, for instance, a fine-grained (Vincent, 2003). They try to take advantages of each parallel approach.

In a first step we analyzed the compatibility of the parallel methods with the steady-sate replacement strategy that used in Bot's algorithm. In a second step, we consider the deployment of this alternative in Open BEAGLE, the GP tool. To use a fine-grained approach we have to create from scratch Bots' algorithm. In the other hand, coarse-grained method can be used within Open BEAGLE with a TCP-IP based communication between

demes. We have selected to apply coarse-grained method to Bot's method. Following subsection describes this approach.

5.2.1 General description of coarse-grained model

In the coarse-grain or island model, the population is divided into small number of sub-populations, called *demes* or *islands,* and each one evolves independently except for occasional migration of copies of individuals between islands (Whitley *et al.,* 1999; Fernandez *et al.,* 2003). This model permits the exploration of different regions of the search space and maintains the diversity of the population thanks to the migration of individuals (Fernandez *et al.,* 2003). The island model requires supplementary parameters in addition to the usual GP parameters (Fernandez *et al.,* 2003):

- Number of sub-populations;
- Subpopulation sizes;
- Frequency of exchange of individuals;
- Number of exchanged individuals;
- Communication topology used.

Optimization of the parameter set to use with the island model becomes a monumental task because they are interdependent and variation of one parameter can change the performance of the overall model (Fernandez *et al.,* 2003). As a result, tuning is done for each parameter at a time (Fernandez *et al.,* 2003) and most of the studies deal with two parameters and leave all others within reasonable ranges (Fernandez *et al.,* 2003; Skolicki and De Jong, 2005): number and type of migrant individuals and frequency of migrations. The other parameters are set before hand. Some of the criteria used are mentioned here. Number of sub-populations or *islands* and number of individuals per island are a trade-off. The number of individuals per island is defined in such a way that the whole population size (as in a *panmictic* population) divided by the number of islands do not reduce too much the diversity inside each island. For instance, population sizes of about 100 individuals in a panmictic version can be split into four or five islands. That generates sub-population of 25 to 20 individuals in each island.

Communication topologies used by migrant individuals are: ring, random, two and three-dimensional meshes. In ring topologies, migrant individuals from island A go to island B and in turn migrants from island B go to island C and so on until migrants from the last island go to the first island A, in a ring topology. In a random communication topology, migrant individuals go to another island randomly selected. In mesh topologies, migrant individuals go to different predefined islands.

Parameters like number and type of migrant individuals and frequency of migrations are studied in GA and GP (Fernandez *et al.,* 2003; Skolicki and De Jong, 2005). Fernandez *et al.* studied the influence of the number of migrants and the frequency of migrations based on exchange of individuals selected randomly. Their study covers some typical GP-problems as "symbolic regression", "ant trail" and "even parity 4" and an application about field programmable gate array design. They found that for a small amount of migrant individuals an exchange rate every generation is the best option. Skolicki *et al.* found in their experiments with mathematical functions which include local optima and a global optimum, that migration interval seems to be a dominant factor while the migration size plays a minor role with regard to the best solution (Skolicki and De Jong, 2005). In addition, they found that migration sizes approaching the population size and large migration intervals degrade performance of the system. They conclude that in general, the best performance is achieved with moderate migration intervals and small migration sizes (Skolicki and De Jong, 2005).

5.2.2 Experiments with parallelization with Island method

The purpose of the experiments with island method is to test if the results are similar or better than those obtained with sequential runs and the use of computational resources. In this case, we have chosen four data sets from those used in Chapter 4. In addition, we tested in a different data set called Ship that comprises 2545 samples with 11 raw features each one, distributed in 8 classes.

We have mentioned that the most important factors to set up a genetic (programming or algorithm) run with the island method are: number of sub-populations, subpopulation sizes, frequency of exchange of individuals, number of exchanged individuals and communication topology used. As mentioned in Section 5.2.1, the number of sub-populations and subpopulation sizes is a trade-off itself that depends on the available resources and can affect the diversity level. In this case, the term diversity only refers to the genetic diversity that large populations have in comparison to small populations (Banzhaf *et al.,* 1998). Even though, in a hypothetical case with unlimited computing resources, we have to consider the amount of individuals in each sub-population, very small populations will reduce the genetic diversity and it could seriously affect the performance of each island. From an initial population of 100 individuals, we could have 4 islands of 25 individuals or 5 islands of 20 individuals.

Since we have at least 5 parameters to experiment with in order to "optimize" the island method applied to Bot's algorithm, we decided to run preliminary tests with parameters set to typical values (adapted to our computing resources) and then adjust parameters accordingly. In addition we had in mind the suggestions from different experiments found in the literature (Skolicki and De Jong, 2005; Fernandez *et al.,* 2003; Cantú-Paz, and Goldberg, 2001; Cantú-Paz, 2007). Initially, we use 5 islands with 50 individuals each. Islands were fully interconnected between them (like a mesh) and migration of 2 individuals to every other island (8 in total) with a frequency of migration of one or two generations. Recognition rate results were slightly lower than results with a single population of 100 individuals. There was not a noticeable difference when using 1 or 2 generations as migrations frequencies. Skolicki and De Jong analyzed the influence of migration size and intervals (Skolicki and De Jong, 2005) and they advise that the frequency of the migrations is more important than the size of migrations. Also they found that migration sizes approaching the population size can drastically drop the performance (Skolicki and De Jong, 2005). In consequence, we reduced the number of migrants to be in the range 10% to 20% of the island population: we choose one migrant to every other island and set the frequency of migration to two generations. This time the performance was very close to runs with a single population of 100 individuals.

Given that each island evolves independently (except for the sporadic migrations), it requires one processor dedicated to the evolution of the island. Therefore, evolutions with five islands will severely impact our limited computing resources (a cluster of 24 machines). So we decided to try evolutions with 3 interconnected islands, each one with a population of about 30 individuals. Migration frequency was set to two generations and migrant individuals were sent to the other two islands.

5.2.2.1 Analysis of results with some UCI data sets

Table 5.2 shows the results of Ionosphere, Pima, Segmentation and Wine (see Table 4.1).

Table 5.2

Recognition rates for some UCI data sets using Island method

Database	**Island**	**Rec. Rate** Ave(%) ± Std	**Evol. Feat** #F*ave* ± **Std(#F)**	**Raw Features** **RawF***ave* ± Std(Raw#F)	**Orig. Raw Feat.**	**Ensemble Rec. Rate (%)** Ave(%) ± Std
Ionosphere	1	81.74 ± 9.38	4.12 ± 2.36	4.21 ± 1.45	34	86.71 ± 8.71
	2	81.14 ± 8.48	4.21 ±2.28	4.37 ± 1.94		
	3	81.06 ± 9.25	4.36 ±2.57	4.33 ± 1.90		
Pima	1	67.94 ± 4.90	3.09 ± 1.14	5.18 ± 1.17	8	70.07 ± 6.13
	2	67.37 ± 5.94	3.17 ± 1.31	5.04 ± 1.37		
	3	67.81 ± 5.21	3.18 ± 1.20	5.17 ± 1.26		
Segment.*	1	90.65 ± 5.69	6.4 ± 2.16	6.59. ± 2.77	19	N/A*
	2	91.46 ± 5.44	4.6 ± 2.03	6.59. ± 2.77		
	3	91.26 ± 6.13	8.8 ± 2.67	6.45 ± 3.08		
Wine	1	83.28± 11.86	3.64 ± 2.15	3.75 ± 1.62	13	89.66 ± 9.70
	2	83.89 ± 11.75	4.20 ± 2.50	3.88 ± 1.66		
	3	81.28 ± 12.94	4.23 ± 2.21	3.72 ± 1.69		

*Since votes of Segmentation data were not recorded, no ensemble results are presented.

The table includes the average recognition rate, the average number of evolved features, average the number of raw features used in the evolved features, for each island. In addition, the table gives the average recognition rate of the ensemble of islands. Columns show results in the format average ± standard deviation. In these experiments ten repetitions have been run each one using 10-fold cross-validation. We use one fold for test, one fold for validation, another for optimization and the remaining 7 folds for training. For each replication this process is repeated 10 times so that optimization, validation and tests blocks are applied to different folds each time. Evolution is run for 10 features (5 for Pima data set) using a *k*-NN classifier and global validation procedure. Results presented take the best individual found in validation and applied to the test set for each fold and afterwards the average is calculated as well as the standard deviation in recognition.

In addition, the individuals to migrate and the individuals to be replaced in the receiving island are randomly selected in order to avoid a big selection pressure that could drive the evolution towards local optima. It is important to mention that during the run of Segmentation data set, the TCP/IP communication link to one of the islands was down for a while. That is why the name of the data set has been signalled with a star in Table 5.2. Even though, evolution of the system continued and there was a gain in the recognition rate. This shows the robustness of the parallelization with the island method.

Table 5.2 shows the average recognition rate for each island. In some cases, they are slightly better than results shown in Table 4.5. The biggest difference resides in the average number of evolved features called #F*ave*. These numbers are greater than those obtained in Table 4.5. That can be caused by two factors: migration of individuals due to island method and global validation. Migration of individuals inserts new genetic material that is useful to continue the evolution towards a global optimum, instead of local optima. That allows the performance to increase instead of being trapped in medium values (local optima). The second factor, global validation, generates a higher number of evolved features because it stops the evolution when there is no gain. If that happens for example after two or three generations of the evolving feature EF4, the solution considered when applying global validation has 5 evolved features

(the last feature, EF5, is the solution at the second or third generation). In contrast, Bot's method rejects the last feature if it does not fulfill the stopping criterion and the final solution only has 4 evolved features.

Bot's method transforms the description of the data set, from an initial space equals to the cardinality of the number of raw features into a smaller cardinality of size #F*ave*. Each evolved feature is an equation in terms of raw features. It is interesting to know how many raw features are used to generate the evolved features. The average number of different raw features and the standard deviation are shown in Table 5.2 as well as the initial number of raw features. It is clear that in all cases, a huge reduction is obtained. Therefore, Bot's algorithm makes not only feature construction but also feature selection at the same time. It uses only the most important raw features to create the evolved features. A deeper analysis about the utilization of raw features is presented in Section 5.4.

Bot's method that inserts new population at each new evolved feature combined with the island method that inserts different individuals (with respect to the individuals already in the receiving island) are a beneficial blend that pushes the evolution to continue. Computation resources were used in a slightly better way because we have used small population sizes. We used panmictic populations of 100 individuals and islands of 30 individuals in three nodes running at the same time. Therefore, the computation time was less than 1/3 the original time required mainly due to the reduction of 10 individuals. In comparison to 100 individuals, we have a reduction of 10% in the number of evaluations (genetic operations and fitness measures).

The island method provides an additional improvement in comparison to results obtained with panmictic populations. Since the islands evolve independently except for with sporadic migrations, they explore different regions of the space, so their solutions can be used at the same time. That is to combine the solutions of each island to form an ensemble. Decision of the ensemble is taken by majority vote.

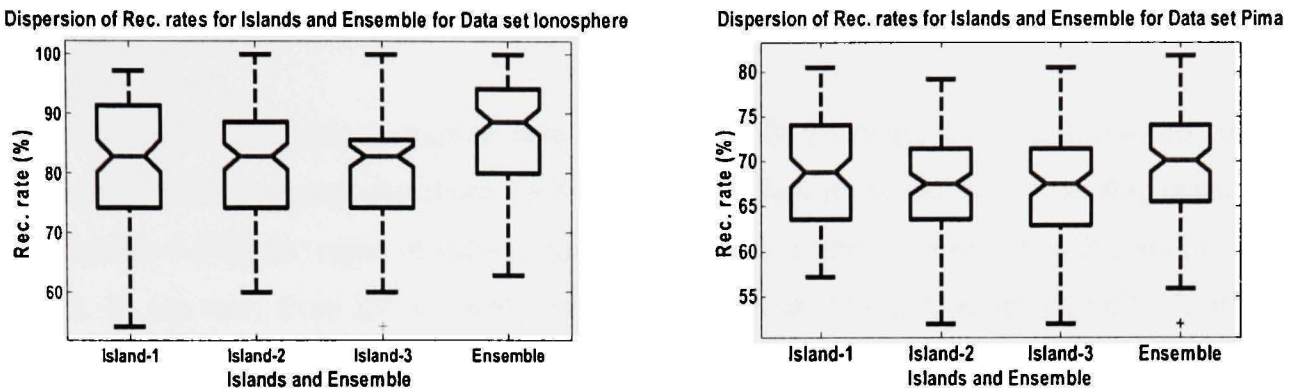


Figure 5.2 ***Dispersion of recognition rates and Ensembles for some UCI data sets.***

Ensemble of three islands provide a considerable gain

To the best of our knowledge, this is the first time that the best result from the islands is not the solution taken, instead the best solution from each island is taken and combined into an ensemble. Figure 5.2 shows the dispersion of recognition rate values for each island, called Island-1, Island-2 and Island-3 and that of the resulting ensemble, for two of the data sets: Ionosphere and Pima. As it is shown, the performance of the built ensembles has a boost in comparison to performances of single islands. In the case of Ionosphere data set, the boost is such that the median value of the ensemble is even better that the third (upper) quartile for two out of three islands. The average recognition rate of the built ensemble is presented in the last column of Table 5.2. We have a gain of about 5% with respect to the average recognition rate of each island. In the case of Pima data set, the improvement is not that big, only 3% in the average recognition rate (see Table 5.2). Figure 5.2 (b) shows that the distribution of recognition rate values of the Pima ensemble is as small gain with a median value greater than 70%. Recognition rate of the ensemble built for the Segmentation data set was not calculated because the votes for some folds and repetitions were not recoded due to the TCP/IP communication mentioned above.

5.2.2.2 Analysis of results with Ship data set

We include the analysis of another data set called FLIR (Forward Looking Infra-Red) that comprises 2545 samples, distributed in 8 classes. This data set is also known as ship because it contains 8 different types of ships. Figure 5.3 presents a sample image of each class in two rows. In top row, from left to right, images of Destroyer (340), Container (455), Civilian freighter (186) and Auxiliary oil replenishment (490). In bottom row, images of Landing assault tanker (348), Frigate (279), Cruiser (239) and Destroyer with guided missiles (208). Numbers in brackets indicate the amount of images in each class. Each object is characterized by 11 raw features: 7 invariant moments and 4 auto-regressive parameters (Rhéaume *et al.*, 2002).

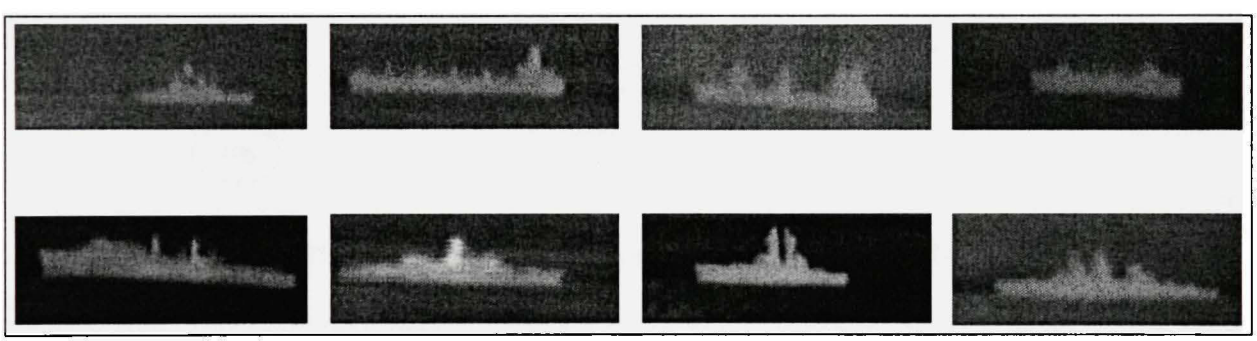
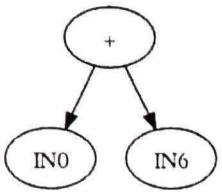
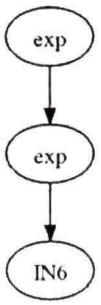


Figure 5.3 ***Samples of images in FLIR data set.***

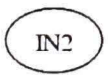
Figure 5.4 shows the typical solutions found using global validation for this data set, in this case for a single fold and repetition. Each tree solution corresponds to one evolved feature. Recall that EF2 and greater depends on the solution found (in optimization) for previous evolved features. In each case we include the tree solution and its fitness in the test set. It can be seen how the recognition rate is increasing along with the addition of evolved features. From evolved feature 4 to 5, there is no increase in the performance: the new evolved feature proposed by the algorithm (EF5) is a different solution than EF4 but has the same performance in the test data set. For the next evolved feature, EF6, the algorithm is able to increase the performance and later it is deteriorated. The stopping point is found based on global validation procedure.



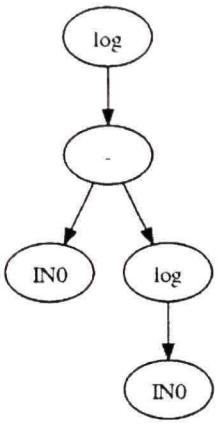
(a) EF1, fit_{TEST}=46.27%



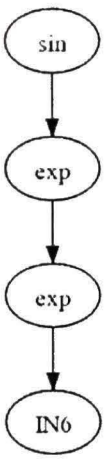
(b) EF2, fit_{TEST}=69.80%



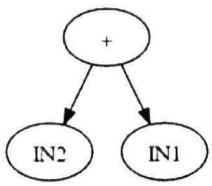
(c) EF3, fit_{TEST}=76.86%



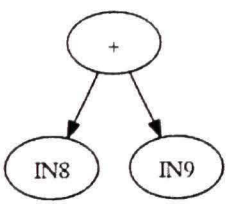
(d) EF4, fit_{TEST}=85.88%



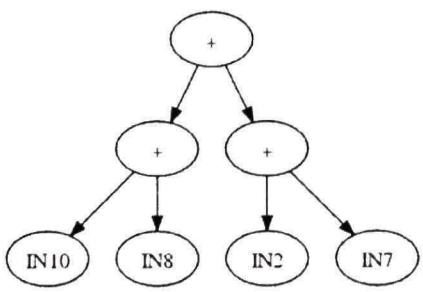
(e) EF5, fit_{TEST}=85.88%



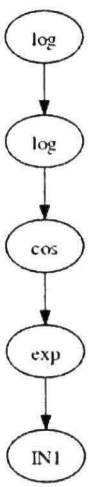
(f) EF6, fit_{TEST}=91.37%



(g) EF7, $fit_{TEST\text{-}7}$=90.58%
EF8, $fit_{TEST\text{-}8}$=89.41%



(h) EF9, fit_{TEST}=91.37%



(i) EF10, fit_{TEST}=95.68%

Figure 5.4 ***Solutions for different evolved features of Ship data set.***

Applying the island method to Ship data set generates encouraging results. Table 5.3 shows the recognition rate and the number of features required when using the island method which generates three different solutions and an ensemble of islands. Results reported in (Rhéaume *et al.*, 2002), (Valin et al., 2006), (Jabeur and Guitouni, 2007) and (Park and Sklansky, 1990) are also presented for comparison purposes. Methods noted as (Rhéaume *et al.*, 2002) and (Valin et al., 2006) are based on the recognition rate for a *k*-NN with k=3 and a distance based on the inverse of the inter-covariance matrix (Rhéaume *et al.*, 2002; Valin et al., 2006).

Table 5.3

Average recognition rates for ship data set for Bot's algorithm with Island method
Average taken over 10 repetitions using 10-fold cross-validation, standard deviation for Rec. Rate and number of evolved features also indicated

Data	Method	Rec. rate (%)		#F*ave*
Ship	Island	Island-1	83.80% ± 5.76	8.03 ± 1.71
		Island-2	82.87% ± 6.55	8.22 ± 1.49
		Island-3	83.23% ± 6.22	8.23 ± 1.52
	Island Ensemble	88.27% ± 2.88		8.16 ± 1.57
	(Rhéaume *et al.*, 2002)	92.88%		N/A
	(Valin *et al.*, 2006)	94.8%		N/A
	(Jabeur and Guitouni, 2007)	89.67%		N/A

Jabeur and Guitouni have used 2 different classification approaches and different selection methods using a holdout method to assess the performance (Jabeur and Guitouni, 2007). They varied the size of the training data set from 30% to 80% of the data set and used the rest of the samples for testing. The best results report a recognition rate of 89.67%. The method used in (Park and Sklansky, 1990) consists in an automated design of linear tree classifiers and is compared to a *k*-NN (*k* varying from 1 to 17) with Euclidean distance metric. The data set is divided in two balanced sets (equal distribution of classes) of approximately the same size. The reported recognition rate with the *k*-NN is 88.3% and it uses the whole set of 11

raw features. Valin *et al.*, made a modification by designing specialized classifiers for each class by selecting the raw features more pertinent. Raw features 1, 6, 9, 10 are very useful classifying classes 1, 2 and 4. Raw feature 11 is important to classify classes 1, 3, 4, 5, 6 and 7 (Valin et al., 2006). Results of Section 5.4 reveal as well the importance of those raw features.

When running the island method, the results are quite similar to those obtained with a single *k*-NN with Euclidean distance. Instead of using 11 raw features, an average of 8.22 evolved features is used to represent the data set with similar recognition rates as shown in Table 5.3 along with the standard deviation. It is important of knowing how many raw features are used to build the evolved features. This analysis is made at the end of this chapter.

Another method presented in Table 5.3 is called Island Ensemble and takes the three solutions found with the island method and combined them to create an ensemble with a majority vote used as combination function. The built ensemble generates an improvement of about 6%. Therefore the solutions provided by the island method are complementary, because each island searches in different regions of the whole searching space. This improvement shows again an additional advantage of using the island method.

Output / Input	1	2	3	4	5	6	7	8
1	**79.5588**	1.2059	0.6765	0.4706	0.9706	5.8529	*8.4118*	2.8529
2	0.2637	**92.1978**	1.4505	2.2198	2.8132	0.2418	0.7912	0.0220
3	1.7112	4.7059	**64.7594**	*19.9465*	7.5936	0.5882	0.3209	0.3743
4	0.1224	1.8776	5.4694	**89.3878**	2.6531	0.3265	0.0408	0.1224
5	0.8908	3.7644	4.0805	4.4828	**82.9598**	2.2126	0.8621	0.7471
6	*6.8929*	0.4643	1.2500	0.4643	3.4286	**78.7857**	1.8214	*6.8929*
7	*9.6250*	2.0000	0.3333	0.3333	1.5833	2.1250	**81.6250**	2.3750
8	2.4286	0.1429	0.3333	0.0476	1.8571	10.3810	0.8571	**83.9524**

Figure 5.5 ***Confusion matrix for classification of FLIR ship images data set.***

In order to have a deeper view of the results we use, in this case, the confusion matrix as shown in Figure 5.5. We can see that for the majority of the 8 classes, recognition rate is between 83.95% and 92.20%. Class 6 (Frigate) is very close with 78.78% and class 3 (Civilian Freighter) seems to be the more difficult with a recognition rate of only 64.75%. In addition, we can mention that class 3 ships are misclassified with class 4 ships (Auxiliary oil replenishment) in a percentage of 19.94%. Other important sources of miss-classification are indicated in italic characters in the confusion matrix in Figure 5.5.

5.3 Over-fit control

The use of EoC helps to increase the recognition rate. Classifiers built are based on a wrapper approach since its response helps to guide the search for new possible solutions of the GP algorithm. It is known that this type of approach and any supervised learning method are prone to over-fitting (Paris *et al.*, 2003). How much the over-fit affects the recognition rate of the evolved solutions is not known *a priori.* We present a classifier based on GP-constructed features without any over-fit control and some strategies to control it. A formal definition of over-fitting presented in (Mitchell, 1997) is:

Definition 1: "*Given a hypothesis space $\mathcal{H}$, a hypothesis $h \in \mathcal{H}$ is said to over fit the training data if there exists some alternative hypothesis $h' \in \mathcal{H}$ such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.*"

It means that the hypothesis has learned (or has been trained) so much the training data, that becomes loosing its ability to generalize to new data. The over-fitting is measured as the difference of recognition rate (or error rate) when using the test data set, between the result with validation after optimization and the result when applying global validation procedure.

5.3.1 Classification without validation

As we have mentioned, the fitness function of the mentioned GP-based feature creator rests on the ability of evolved features to classify data for a given problem. Once the evolutionary system has been evolved using the optimization data set, performance has to be measured to know how well the system classifies unseen data of the problem in hand. If the data set (or part of it) used during optimization is also applied to measure the system performance, results are biased because the system is measured with the same data it was optimized. Therefore, results generated by a classification algorithm without a validation strategy are not a real estimate of the generalization power of the classification algorithm.

5.3.2 Classification with validation after optimization

As a standard procedure, validation is carried out in pattern recognition and data mining field. Validation is carried out once the system is optimized. The solutions proposed by the system at the end of the optimization process are corroborated (validate the model) by applying them to the validation data set, i.e. only the last population is validated. As a result, validation finds the best solution (among the ones obtained during optimization). The solution that performs best with this set is chosen as the final solution. The general algorithm of Bot's method with validation at the end is presented in Annex III.

The utility of an algorithm resides in its power to classify new unseen data based on the training already acquired. The ability of generalization can be measured as the recognition rate of the algorithm when it is applied to new unseen data. This dataset is the test set. Performance of the GP-based system is calculated by applying the best solution of the last population to the test set.

5.3.3 Global validation

It has been shown that even with the validation phase (after optimization) results are over fitted towards the set used during the optimization phase (Radtke *et al.,* 2006; Dos Santos *et al.,* 2006). They projected the best solution found with a GA and the Pareto-front found with NSGA-II algorithm into the validation set and showed that best individuals in optimization set had not the same performance in the validation set. There were other solutions that perform better. This means that during the optimization process, solutions with good generalization power were detected but not considered further. These solutions were not identified as consideration was only given to best individuals at the end of the optimization process.

To avoid that, a strategy called Global Validation (Radtke *et al.,* 2006), has to be set. It measures the performance of the whole evolving population against the validation set, ranks the individuals by performance and records the best evolved candidate measured in the validation dataset for each generation in a file called the *auxiliary archive* S. The approach consists in verifying after each generation, the performance of all the evolved candidates found along the optimization process i.e., the whole population, but measured over the validation dataset. Evolved candidates are ranked by fitness measure (recognition rate) on the validation dataset and the best evolved candidate found in generation g is compared to the best evolved candidate found in the previous generation (g-1). If the evolved candidate at generation g performs better than the solution retained in generation (g-1), the auxiliary archive S is updated. Otherwise, solution at generation (g-1) is retained. In the latter case, when the performance on the validation set does not increase due to new solutions, we have reached the stop point. Therefore, logging in every solution is a memory of the evolution process and the archive S permits to detect when over fitting is produced. Results obtained in the auxiliary archive do not modify in any way the curse of the evolution; they are only an external measure of performance of evolved individuals in the validation set. Annex V shows the pseudo-code the global validation strategy.

5.3.4 Global validation applied to Bot's procedure

In the global validation method, validation is done at every generation concurrently with the optimization phase. Following lines explain how to integrate the global validation strategy within the Bot's algorithm and the detailed pseudo-code is presented in Annex V.

At the beginning (feature f=1 and generation g=0) an initial random population is created, fitness is measured for each individual in the population, with the optimization data set and with the validation data set. The individual with highest fitness measure in the validation data is copied along with its fitness into the auxiliary archive S. This is the first individual in the auxiliary archive. For subsequent generations, update of the auxiliary archive S works as follows (see Figure 5.6). Suppose that i-1 "definite evolved features" have been already defined, that is EF(1), EF(2),..., EF(f=i-1), with ($1 \leq i \leq Max\_Nb\_features$), GP is running to create evolved feature i and the current generation is g. The additional steps that global validation introduces to Bot's algorithm (presented in Section 4.1) are: the projection of evolved individuals into the validation dataset, fitness measure for the projected population, ranking by fitness and auxiliary archive update. These supplementary steps are executed for each generation and each evolved feature. The other steps in Bot's method are executed as the standard procedure. The fitness calculation (explained in Section 4.1.2) for each individual in population P(f=i, g) projected over the validation data, considers the individual of the current generation g juxtaposed (indicated as $\oplus$) to the definitive evolved features EF(1), EF(2),..., EF(f=i-1). Individuals are ranked by fitness and the best individual (in validation) is compared to the previous best individual in validation for feature i-1 and generation (g-1). If fitness is higher, the auxiliary archive S is updated with this new individual along with the fitness value, the generation g to build the feature i. Otherwise the auxiliary archive S is not modified and the previous individual (in validation) continues having the best fitness. It is worth to point out that the definitive evolved features EF(1), EF(2), ..., EF(f=i-1) are the output from the optimization process. They are defined as $EF_{prior}(f\text{-}1)$ in Annex V.

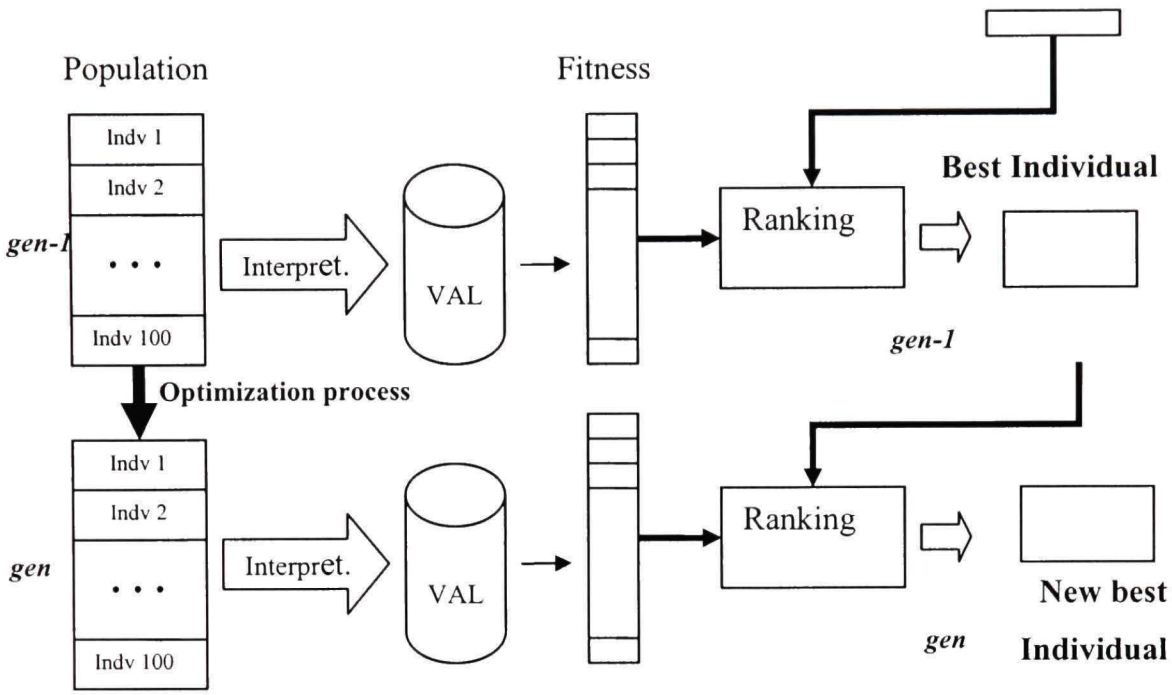


Figure 5.6 ***Global validation procedure.***

Individuals from generation g are interpreted on the validation data set. They are ranked by fitness value along with the best individual from generation (g-1), stored in the auxiliary archive S. The individual with highest fitness value is selected and archive S updated accordingly. Same comparison is made for all generations

To perform the validation phase, the optimization dataset is changed by the validation dataset for each generation. Once the auxiliary archive S is updated (or not), the optimization data set is set back to continue the optimization process as normal.The additional steps mentioned below are repeated for each generation. Once GP has run for *Nbr_Gen* generations, Bot's stopping criterion is evaluated to decide if the selected best after *Nbr_Gen* becomes the evolved feature EF(*i*). This stop criterion depends on the incremental recognition rate gain that would be generated by the addition of the current best individual juxtaposed to the already evolved features and is calculated over the optimization dataset. GP algorithm is run for *Nbr_Gen* generations or until the stop criterion is met. Once GP has run for *Nbr_Gen* generations, it starts the process for the next feature and this is repeated until we have *Max_Nb_Feature* features or the Bot's stopping criterion is met. The resultant individual from the auxiliary archive S, is selected to calculate the final recognition rate with the test data set.

5.3.5 Analysis of over-fitting and stopping criterion

At the end of the optimization and validation procedures, two solutions are retained: the individual with highest recognition rate in the last generation when using the optimization data set and the individual with highest recognition rate on the validation data set (last individual in the auxiliary archive S). Along with the individuals and their fitness values, the feature and generation numbers from where these solutions came, are also recorded. When comparing these two values of recognition rate, we only have two possible outcomes: the solution in the auxiliary archive S has a better recognition rate or the two recognition rate values are equal. In the first case, there is over-fitting and the global validation strategy was able to detect it. Furthermore, the global validation strategy detected the moment that over-fitting started, because the evolved feature and generation values were recorded. In the second case, the two individuals have the same fitness. After fitness comparison, tree size and depth are evaluated in this order to reduce the possibility that different individuals are taken as identical. If individuals are different in structure but with the same fitness, tree size and depth, we consider the two of them are equivalent (in performance) and we make no preference for one or the other. Structural components are not verified during comparison because we are focused in performance and the two individuals are equivalent from this comparison point and, in addition, a structural comparison will increase unnecessarily the validation time.

As mentioned before, the global validation strategy helps to identify the right moment to stop the evolution process and avoid or at least control the effects of over-fitting. For instance, in standard GP runs a maximum number of generations for evolution are set *a priori* before starting the evolution. If the number of generations in which the best solution was found within the auxiliary archive is very small in comparison to the number of generations used in the GP evolution, we know that evolution could be stopped before. From this point further, the evolution has over fitted the optimization data set, generating solutions with less generalization power. On the other side, if the two values of number of generations are similar, we have identified that the stopping criterion (set in terms of number of generations)

was well defined and the global validation strategy have found the best individual of the evolution.

Application of a stopping criterion based on global validation strategy is effective to control the over-fitting as shown before. But also it can be applied to Bot's algorithm to stop the addition of new evolved features in a more precise fashion. To explain this improvement, we have to analyse in more detail the already involved stopping criteria. In Section 4.1 we referred to two different stopping criteria (see algorithm 2). The first one, called evolution stopping criterion (Evol_stop) makes reference to the classical stopping criterion of evolutionary algorithms. In general, Evol_stop criterion is met when an acceptable error rate (or recognition rate) level is obtained or when the maximum number of generations (defined *a priori*) is reached. In practice, more often the second condition is used because expected error rate is normally not attained within a reasonable number of generations. Evol_stop criterion is evaluated for each generation of the evolution.

The second stopping criterion, called Bot_stopping identifies the moment where is not advantageous to have an additional evolved feature because the increment generated in performance is inferior to the threshold d (defined in equation 4.2). As a result, this new evolved feature is discarded. Bot_stopping criterion is evaluated after *Nbr_Gen* generations.

Setting the conditions of Evol_stop criterion too high (setting a large number of generations or a 100% of recognition rate on optimization) could lead to over fit the optimization data. In turn, Bot_stopping waits for *Nbr_Gen* generations before being evaluated and then takes the decision about stopping. The right moment to stop can be decided in a better way by using the global validation method because it keeps watching the solutions during evolution (optimization) and their performance in validation. If the stopping point triggered by the global validation occurred before completing the *Nbr_Gen* generations, the proposed best solution can be also evaluated with the Bot_stopping criterion to decide if it is accepted or not as the last definitive evolved feature. In this case, we have not waited until *Nbr_Gen* generations and the proposed best solution does not over fit the optimization data. In brief,

the global validation strategy provides a mean to control the over-fitting and also it can complement the Bot's stopping criterion.

There are two points to mention respect the original Bot_stopping criterion. First, the evaluation is done at the end of evolution for each feature. The criterion decides if the feature is accepted or not. It means that no evaluation is done for the generations 0 to 10 (11 is the maximum number of generations). Consequently Bot's stopping criterion can be assimilated to validation at the end of the evolution. It is not exactly equivalent because the verification is done for every evolved feature. As a second remark, we have to consider that if at any evolved feature the relative increment in recognition rate is not greater than the threshold *d*, this last evolved feature is discarded. As a result, it is less likely that performance values have going into over-fitting or at most have a small amount. The prompt discarded of the feature reduces the chances of accepting over-fitting solutions.

We believe that Bot's procedure provides a validation level better than validation at the end of the evolution and it minimizes the occurrence of over-fitting by eliminating the whole last feature without considering possible improvement achieved in between features.

5.3.6 Experiments to analyze the over-fitting

The results reported in Chapter 4, were based on the stopping criterion used in Bot's algorithm and 10-fold cross-validation. In this section, we apply the global validation procedure to some of the UCI problems analyzed in Chapter 4 and compare the results. Recognition rates reported in Chapter 4 were the average of the 10 folds calculated in validation. We have taken the Pima data set (see Table 4.4 for a description of the data set) to apply the global validation procedure.

5.3.6.1 Comparison of results using Bot's procedure and Global validation

To show the differences between Bot's stopping criterion and the stopping criterion according to the global validation procedure, we will use one fold. As explained in Sections 4.1 and 5.3, Bot's stopping criterion is based on the comparison of the relative gain in recognition rate in optimization, calculated at the feature where the recognition in validation is maximal against the threshold d (see equation 4.2). Figure 5.6 shows the curves of recognition rate for optimization and validation and test in solid lines. The maximal recognition rate in validation is attained at feature 3. Therefore, the threshold d for this single experiment is calculated as:

$$d = \frac{t_i - t_{i-1}}{t_{i-1}} = \frac{87{,}00 - 86{,}91}{86{,}91} = 0{,}001035, \tag{5.4}$$

We compare if the relative gain in optimization for each additional feature is greater than the threshold d. So, this comparison is made from feature 2 to 5 (maximal value). If the relative gain in optimization (see equation 5.3 and in Figure 5.6, the blue solid line) is greater than the threshold d, the feature i is accepted; otherwise, feature i is discarded and the final number of evolved features are from 1 to i-1. It is worth to mention that threshold d is estimated over all the data set. In this case, we have taken the corresponding value from one fold to explain how it is calculated. Figure 5.6 shows different curves for the case of one fold. In this particular case, it is clear that evolved features 1, 2 and 3 are accepted and feature 4 is rejected, because there is no relative gain as the recognition rate in optimization does not increase from the 3rd to the 4th features (see Figure 5.6). The final solution is then:

$$Def\_Sol = EF_1 \oplus EF_2 \oplus EF_3, \tag{5.5}$$

Figure 5.7 includes additional information. The blue curve with circles as markers corresponds to the best individuals in optimization taken from the last generation of each feature (Bot, 2001). The maximal recognition rate in optimization is 87.00% is obtained at feature 3 and remains the same value thereafter (note that there is a very small improvement from feature 2 to feature 3; it is difficult to see because of the scale of the figure).

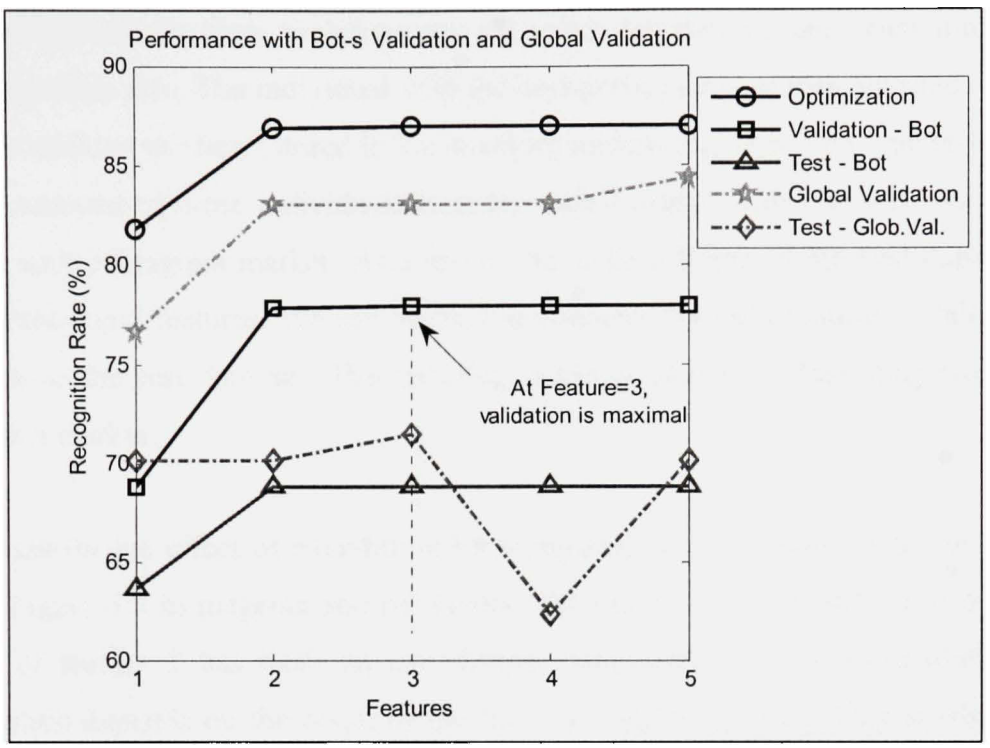


Figure 5.7 ***Comparison of validation with Bot's method and Global Validation.***

The validation curve with Bot's method is calculated as follows: in the last generation of each feature, the best individual in optimization is mapped into the validation data set. This curve is shown in black color with square markers. The maximal recognition rate is 77.91% at feature 3 and remains in that value (note as well here that there is a very small improvement from feature 2 to feature 3, which is difficult to see because of the scale of the figure). Figure 5.7 includes as well a solid red line with triangular markers. This line corresponds to the test results, which are the mapping of F_{SOL} into the test data set. We see that the maximal recognition rate attained is 68.83% and it is already reached at feature 2. These are the result obtained for a single fold of Pima data set using the Bot's procedure and its corresponding stopping criterion.

If we use global validation, we keep one eye in the individuals that generalize better which are stored in the auxiliary archive S. The last individual in this archive is the one to be use to calculate the recognition rate on the test data set. The validation curve generated with the global validation procedure takes the very best individual in validation. This is done by

mapping all individuals from evolution into the validation data set and calculating for each one its recognition rate. The individual with the best performance is then selected. This is the one that generalizes the best (stored in the auxiliary archive S). Then new (global) validation curve is composed of those individuals (one for each feature). This curve is shown in dash green line with pentagram marker. As a result, the archive S has all the best individuals for each generation and features. We use the best individuals for each feature in validation and apply them to the test data set. The resulting curve is shown in dash magenta line with diamond as a marker.

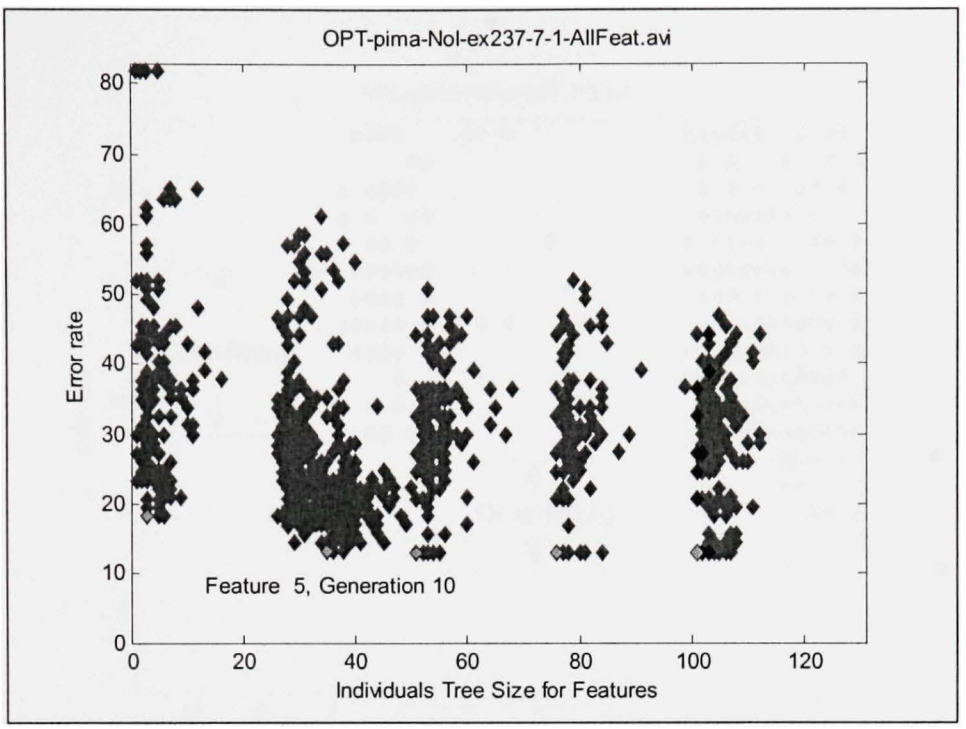
We can quantify the effect of over-fitting by comparing the recognition rates in test of the curves in Figure 5.7 in magenta and red colors. For feature 1, the difference is 6.49%. The evolution of feature 2 has itself an over-fitting component because the evolution of the second feature depends on the result of the first one (Bot's method). That is why the final solution is expressed always as shown in equation 5.5. Since the global validation curve (green color in Figure 5.7) increases its recognition rate, we could take it as part of the solution. For that case, the over-fitting calculated in test is reduced to 1.29%. Then we can continue calculating the impact of over-fitting. Based only in the validation curve created with the global validation procedure (green curve) we can continue until feature 5, but we see that the mapping into the test set has a decline from feature 3 to 4. Where the right point to stop is? It is not easy to establish it based only on the validation curves at the end of each feature. We have to analyze what happens during the evolution of each feature. We can do that with the global validation because it keeps on eye in the generalization power for the population through every generation. This will be explained in the following subsection.

5.3.6.2 Analysis of over-fitting with global validation:

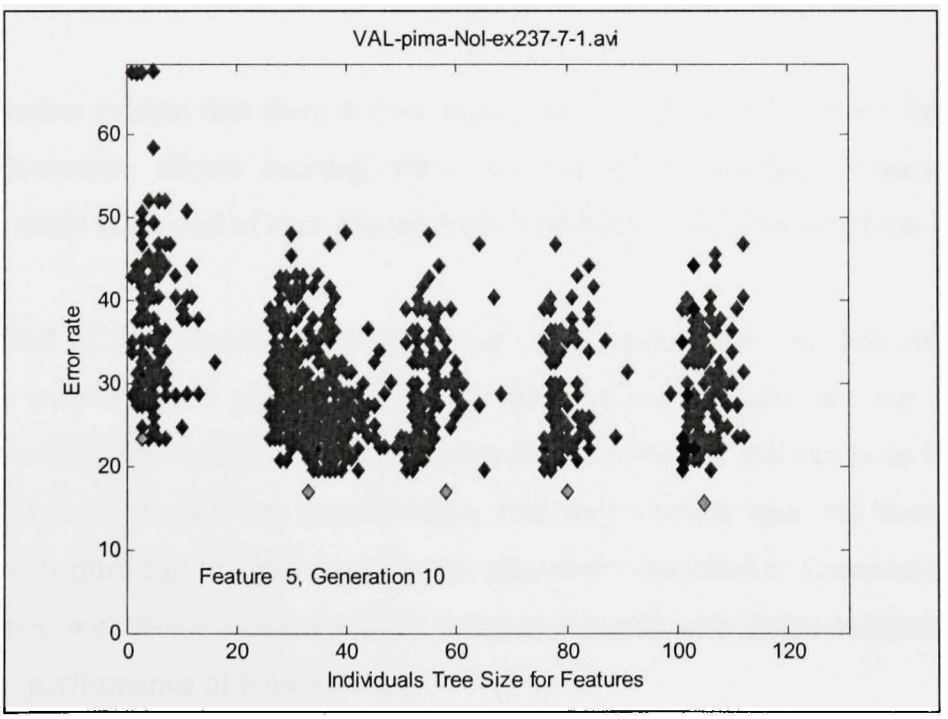
When global validation is applied, we mapped every individual generated during the evolution into the validation data set and find the best. We start doing that since the first generation in feature 1. If the best individual in validation is better in subsequent generations (and features) the auxiliary archive S is updated accordingly. However, if at any point the

best individual in validation is worse than the recorded, we conserve the previous best individual (from validation). Note that best individual means here the individuals with highest recognition rate and the opposing for worse individuals.

With the mapping of individuals generated in optimization into the validation data set, we keep an eye on individuals to see whether they are also the best in validation. Figure 5.8(a) shows the final status of the evolution. The figure shows the error rate (= 100% - recognition rate (%)) for each generation and feature. Each red diamond represents one individual of the population. We can see five different clouds of points corresponding to five features in the evolution. For each evolved feature the cloud is more or less wide, according to the individuals' sizes (tree sizes). This allows us to show the diversity of individuals in the same generation or feature. In each cloud there is a green diamond that corresponds to the best individual in optimization. Dots in blue correspond to the individuals in the current generation. As indicated before, this is the final status of the evolution (feature 5 and generation 10). In Figure 5.8(b) we show the mapping of all individuals from evolution into the validation data set. Again, it is shown the error rate for each feature and the width of the clouds correspond to individuals' sizes. In this case the green diamond are the best individuals in validation for each feature and the smaller black dots correspond to the mapping of best individuals of optimization into this data set (validation). Hence, we can see that best individuals in optimization are not necessarily the best in validation. It means that these individuals (best in optimization) are not the best individuals to generalize. Only the best individuals in validation and optimization for each feature are displayed in different colors. Figure 5.9 shows some details from Figure 5.8(b). It illustrates the occurrence of over-fitting for features 1 and 2 (shown in the figure): the difference in recognition rate between the best individuals in validation (green dot) and the best individual in optimization mapped into validation (black point).



(a) Optimization



(b) Validation

Figure 5.8 ***Searching space in optimization (evolution) and mapping into validation.***

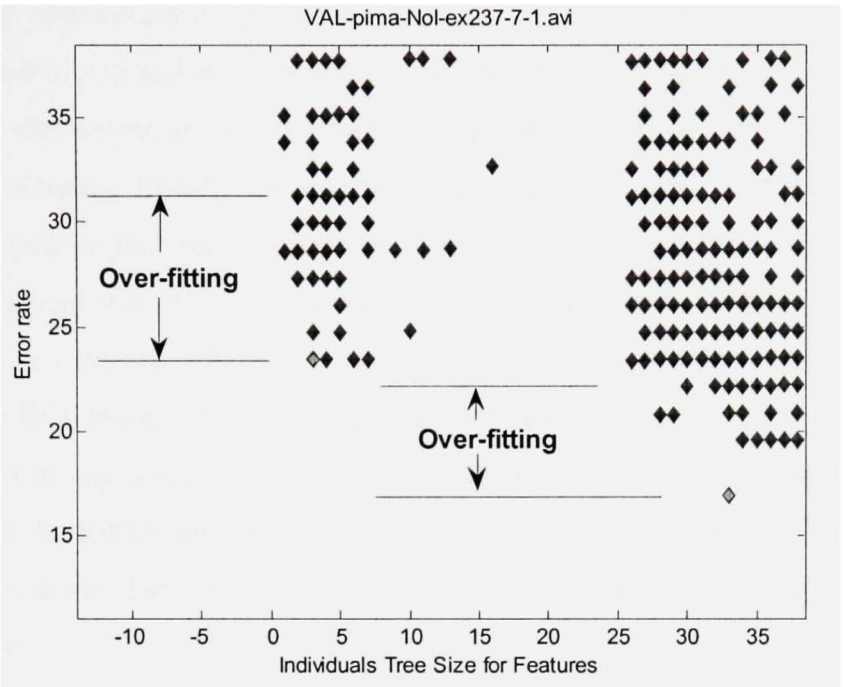


Figure 5.9 ***Zoom of Figure 5.3(b) showing over-fitting for features 1 and 2.***

Figure 5.9 makes evident that there is over-fitting, but this figure only shows that the global validation procedure allows locating when the over-fitting appears, because the same comparison made at the end of each feature is done for each generation in global validation.

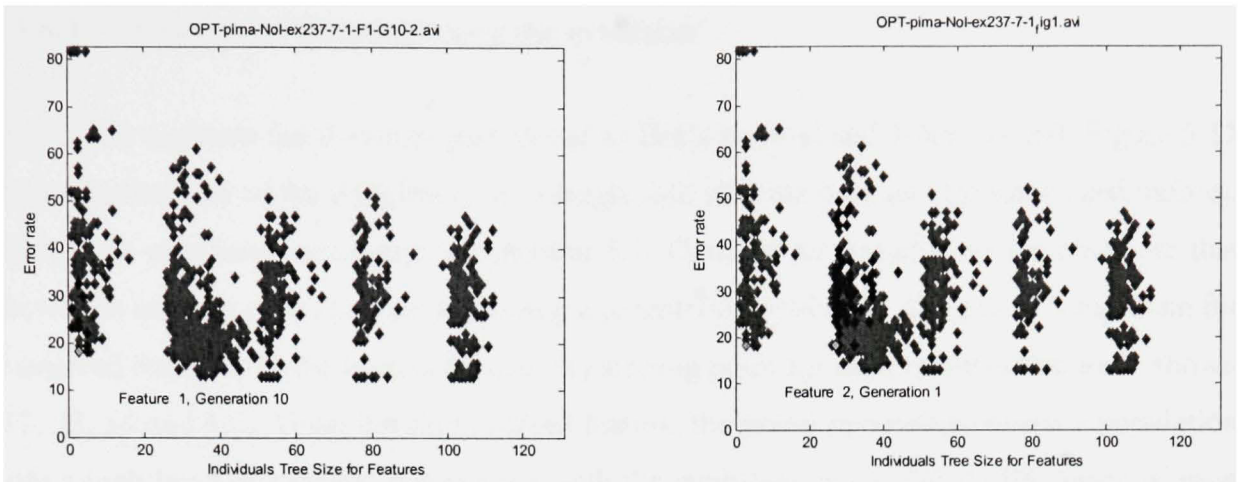
The true effect of this generalization power has to be seen in the test data set. As Figure 5.8(b) show over-fitting is produced since the first evolved feature. We can quantify the effect of over-fitting by comparing the recognition rates in test of the curves in Figure 5.7 in magenta and red colors as it was shown before. For this particular case, we have over-fitting from the first feature and the global validation procedure identified it. Comparing the values in test obtained with Bot's solution and the solution selected with global validation, we see a difference in performance of 6.49%.

Global validation showed that, in some cases, over-fitting can appear early in the evolution towards the first feature, as shown in figures 5.8 and 5.9. Given that in Bot's method, the

evolved features greater than the first one depend on all previous evolved features, what is the influence of over-fitting on the whole solution? As shown in Figure 5.9, this influence varies from one evolved feature to another. The amount of over-fitting in the validation data set decreases in the second evolved feature (see Figure 5.9). The evolution from one evolved feature to the following, founds new solutions that generalize better, and they are recorded as soon as they appear in the auxiliary archive. This is also shown because the recognition rate obtained with global validation in the test set (dash-dot line in magenta in Figure 5.7) is better than the red line (solid) that remains in the same recognition rate level. Even, for evolved feature EF3, there is an increase in the performance. Therefore, we can say that over-fitting produced at any point in the evolution makes subsequent solutions have an implicit over-fitting that is controlled and in some cases reduced, because the algorithm permits finding new solutions that generalize better. This is consequence of the evolution of one feature at a time.

5.3.6.3 Analysis of evolution

Displaying the population during optimization and validation phases gives some insight about the evolution and global validation process. The information related to over-fitting provided by Figures 5.8 and 5.9 was analyzed in previous subsection. In this part, we take snapshots of the evolution at different stages in Figure 5.10. Part (a) shows the status of the evolution at the last generation (10) of the first evolved feature EF(1). In Figure 5.10 the error rate is displayed for 5 evolved features used during the evolution. The red diamonds represent the whole population. As mentioned before, population for each cloud represents the population for each evolved feature. Each individual is set place in a location according to its error rate (vertical axis) and its tree size (horizontal axis). The current population at generation 10 of feature 1 is represented with blue diamonds. Green spots correspond to the best individuals in optimization at that stage and before (feature 2 and 1 in part (b)).



(a) Evolution, feature 1, generation 10 (b) Evolution, feature 2, generation 1

Figure 5.10 ***Distribution of the population in evolution at two different stages.***

We can see in part (a) that there were individuals with very high error level (around 80%). Population for the first feature is well distributed mainly from 18% to 65% error. The last population in feature 1 is concentrated in the lower part of the first cloud, which means that the evolution of individuals has reduced the error rate. Starting the evolution for feature 2 (see Figure 5.10 (b)), population is again randomly initiated. That explains why there are blue diamonds with error rates from 20% to 50%, which is greater than last population of feature 1. The re-initiation of the population for each new feature inserts new and different individuals, so that diversity is injected. Given that the error rate of individuals in any feature (greater than 1) depends on the results from previous features, after 1 or 2 generations in the new feature, the best individuals attain the same error level or even smaller. See, for instance, that the green spot for evolved feature 2 has the same error rate than the best individuals in the last generation of previous feature. The number of generations for each feature allows population to progress in the standard way and the sudden change of population for each new feature avoids having a population composed of copies of just a few good individuals. This helps to prevent stagnation of the population to local minima. As shown in Figure 5.10, the specific data set (Pima) only allows a small reduction of the error rate through the evolved features.

5.3.6.4 Analysis of diversity along the evolution

In order to evaluate the diversity gained due to Bot's method and Island model, Figure 5.11 shows the entropy of the population for a single fold of Pima data set (the same used before). Entropy is calculated according to equation 5.1. Concentrate initially on the blue line that shows the entropy of the population during a panmictic population. The run is being done for 5 evolved features. In the horizontal axe, the starting point for each evolved feature is shown (12, 23, 34 and 45). Along the first evolved feature, the initial population inserts a population with a high level of entropy. It decreases with the evolution, but at the starting point of each new feature, there is a new initial population, so entropy suddenly rises to levels comparable to the generation 0 and evolved feature 1. Again, during the evolution, entropy decreases. As it can be seen the loss of diversity is restored thanks to the re-starting random population of Bot's method for each new evolved feature without loosing what has being gain (new evolved features take into account the previous evolved feature).

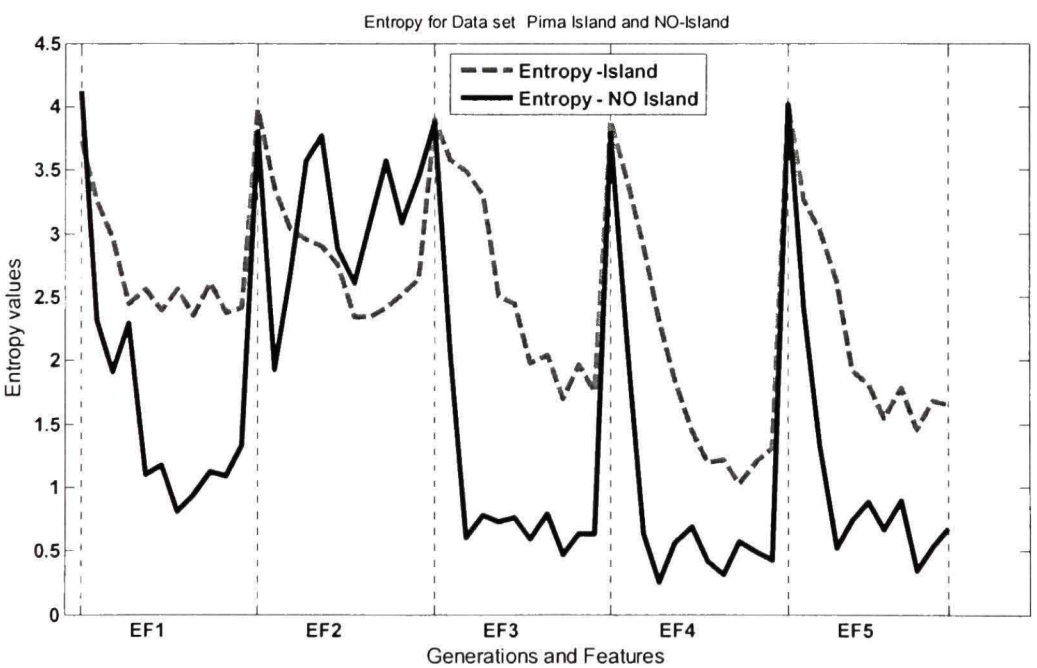


Figure 5.11 ***Entropy as a measure of diversity during the evolution. Island and panmictic.***

The dotted red line in Figure 5.11 shows the entropy levels when the Island method is used. It is clear that, an additional diversity is injected due to the migrants entering each island (each two generations). The levels of entropy (diversity) are superior along all the evolution. This considerable gain in diversity is due to the different regions that are searched by each island.

5.3.6.5 Review of results

We have applied the global validation procedure to five data sets used in Chapter 4: Ecoli, German, Ionosphere, Iris and Pima. We have used 10 fold cross-validation and five repetitions with and without global validation. Data sets have been partitioned in 10 non-overlapping blocks called folds of approximately the same size. One block for optimization, one block for validation, another block for test and the remaining 7 blocks for training. Section 4.4.1 includes additional details concerning the data set distribution. Table 5.4 presents the average recognition rate on test data and average number of features when using a k-NN classifier with k=3. Table 5.5 shows the results when using an MDM classifier. In both cases, we obtain better results applying the global validation procedure in three out of five data sets, but the differences are not significant. It can be seen as well that the average number of features with global validation is greater in all cases but Iris data set. That corresponds to the concept of early stopping used in Bot's algorithm to avoid over-fitting.

The Bot's stopping criterion determines if is useful to add additional features according to the gain in recognition rate (in optimization). It considers only the final results at the end of a feature. In contrast, global validation keeps looking the performance for every generation, so it is able to detect over-fitting even before completing all the generations for the first evolved feature. Iris data set is an example of this case. Only one evolved feature is required. We observe as well that results with k-NN classifier are, in general, better than those with MDM classifier but differences are not that big.

Table 5.4

Comparison of results when applying global validation to Bot's method with *k*-NN
Five repetitions using 10-fold cross-validation, average and standard deviation values included for recognition rate and number of evolved features

Database	Results Bot's method without global validation		Results Bot's method with global validation	
	Rec. rate (%)	#F*ave*	Rec. rate (%)	#F*ave*
Ecoli	76.82 ± 7.24	2.7 ± 1.37	75.64 ± 8.81	3.19 ± 0.98
German	70.30 ± 3.49	1.9 ± 1.15	70.58 ±6.03	2.09 ± 1.37
Ionosphere	80.45 ± 8.78	1.4 ± 1.83	82.97 ± 8.71	3.68 ± 1.52
Iris	94.40 ± 5.26	1.8 ± 0.83	93.60 ± 7.84	1.06 ± 0.9
Pima	67.14 ± 7.34	1.75 ±1.37	68.13 ± 4.5	2.32 ± 1.39

Table 5.5

Comparison of results when applying global validation to Bot's method with MDM
Five repetitions using 10-fold cross-validation, average and standard deviation values included for recognition rate and number of evolved features

Database	Results Bot's method without global validation		Results Bot's method with global validation	
	Rec. rate (%)	#F*ave*	Rec. rate (%)	#F*ave*
Ecoli	70.35 ± 7.94	3.1 ± 1.06	66.17 ± 11.67	2.94 ± .097
German	66.84 ± 5.33	1.3 ± 1.88	66.30 ± 5.63	1.25 ± 1.39
Ionosphere	75.42 ± 10.99	1.1 ± 1.85	76.28 ± 12.51	1.88 ± 1.38
Iris	94.00 ± 6.76	1 ± 0.88	94.40 ± 6.50	1.02± 0.83
Pima	69.53 ± 7.60	1.07 ± 1.45	69.63 ± 5.93	1.79 ± 1.30

5.4 Feature selection and feature creation

Bot's method constructs evolved features using GP as evolutionary tool. Each evolved feature is a formula of the terminal and function sets. That is, Bot's method is a feature creation strategy. The purpose of the feature creation is two-fold: to transform the initial raw feature space into a reduced space of evolved features and to produce classification rates of the same level of performance than those attained with the initial space of raw features. In brief, we can say that Bot's method proposed to create a compact description of a problem that yields similar or better performance than the initial representation. The initial representation is the set of raw features that describe the problem. For instance, Ionosphere dataset (from UCI) is represented with 24 raw features (see Table 4.1). Bot's generates a compact description of 6 evolved features (in average). A *k*-NN classifier on the normalized data with a cardinality of 34 produces a recognition rate of 74.9% (see Table 4.5) meanwhile a compact representation with 6 evolved features generates an average recognition rate of 81.74%. If the ensemble of islands is used, the representation uses less than 15 evolved features and attains an average of 86.71% in recognition rate. This example illustrates the advantages provided by Bot's method.

Since each evolved feature is a formula of the terminal and function sets, it is important to know from the initial raw features, how many are used to build the evolved features. If, in average, all the raw feature, Bot's method generates a transformation into a compact description that produces similar or better performance results. Only this constitutes a positive outcome as it has been shown before. But what about if only some of the raw features are consistently used to create the evolved features? This means that Bot's procedure is a feature creation and, at the same time, a feature selection process. This section performs an analysis of the utilization of raw features for the data sets used in this chapter.

GP manages a population of individuals; each of them is a solution to the problem at hand. Individuals in the population are formulas where the operators (+, -, *, /, etc.) are the function set used and the variables are the terminal set (raw features of the problem description). Raw

feature utilization can be analyzed from different standpoints. In one side, we can analyze how the utilization of raw features is through the evolution. In another side, we can analyze which are the raw features that are part of the final solutions. In the former case, there could be raw features frequently used during the evolution that are not part of the final solution. In the latter case, the participation of the raw features during the evolution is not considered even if they important to generate the final solution.

5.4.1 Analysis of raw feature utilization during evolution

To analyze the raw feature utilization through the evolution we take the evolution during a particular fold and replication. Figure 5.12 shows the utilization of all raw features through the evolution for the dataset Ship. Evolution is run for 10 evolved features and each one is evolved during 11 generations. Ship data set is described by 11 raw features, each of them presented in Figure 5.12. Each curve reflects how many times the raw feature has been used during the evolution (generations and evolved features). Raw feature 1, shown in blue, was very used during evolution of evolved feature EF3 (it reaches 80 times). Also, it is clear its utilization during evolution for EF4, EF6, EF7 and EF9. The same analysis can be done with other raw features. In general, this figure reveals the importance of each raw feature during the evolution.

The horizontal line placed at a height of 11 corresponds to the 11 raw features of the data set. It is depicted to make easier the comparisons. Most of the curves are at low level of utilization and only some of them overpass the 11 level. The red thick curve is the average number of raw features used through the evolution. Therefore, in average, less than 11 raw features are used during the evolution. Only for a few cases, the average overpasses or is close to 11. This figure gives an idea of the real number of raw features that are required during the evolution. It shows clearly that Bot's method is at the same time a feature creation and feature selection method.

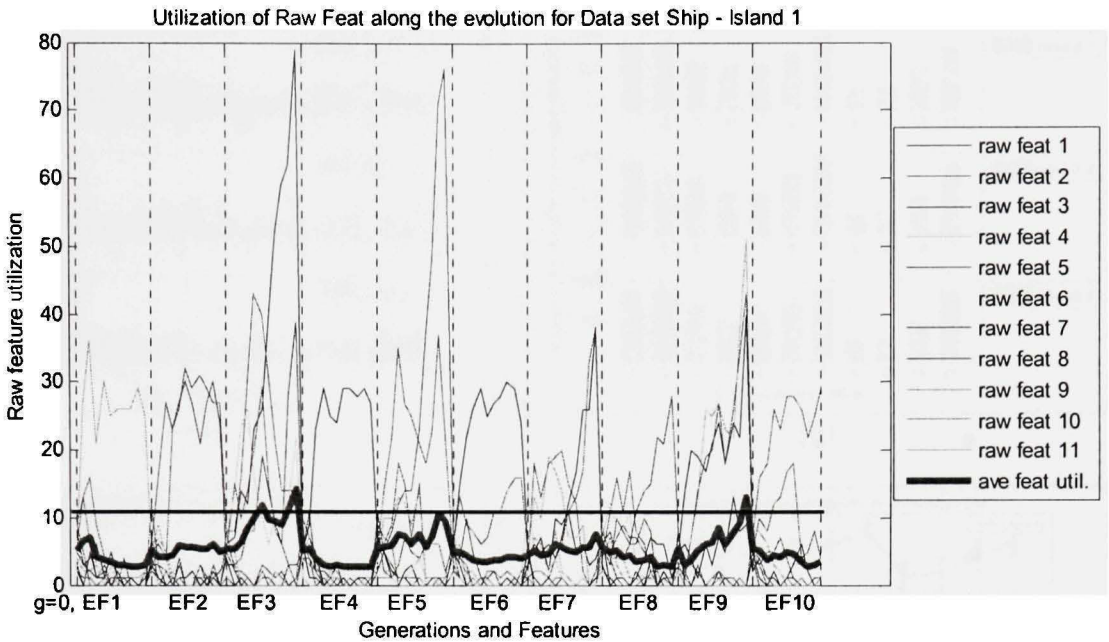


Figure 5.12 ***Utilization of raw features through the evolution- Data set Ship.***

5.4.2 Raw features utilization in evolved features

The other approach mentioned before is an analysis of the frequency of utilization of raw features in the final solutions. Figure 5.13 (a) and (b) show the average of utilization of raw features to build the evolved features for two data sets: Ionosphere and Ship. Average is taken over 10 repetitions of 10 folds of the data set for each island and is presented as percentage (100 in the scale if the raw feature is used in all repetitions and folds). In the case of Ionosphere in (a), raw features 5 and 6 are the most frequently used in the evolved features, about 40% of cases. For Ship data set in (b), there are 5 or 6 raw features used more than 50% of the cases. Raw features 1, 2, 7 and 11 are use in most of the cases. We can say that in the Ionosphere case is clear the importance of a few raw features in the final solution whereas in Ship case, most of the raw features take part in the final solutions.

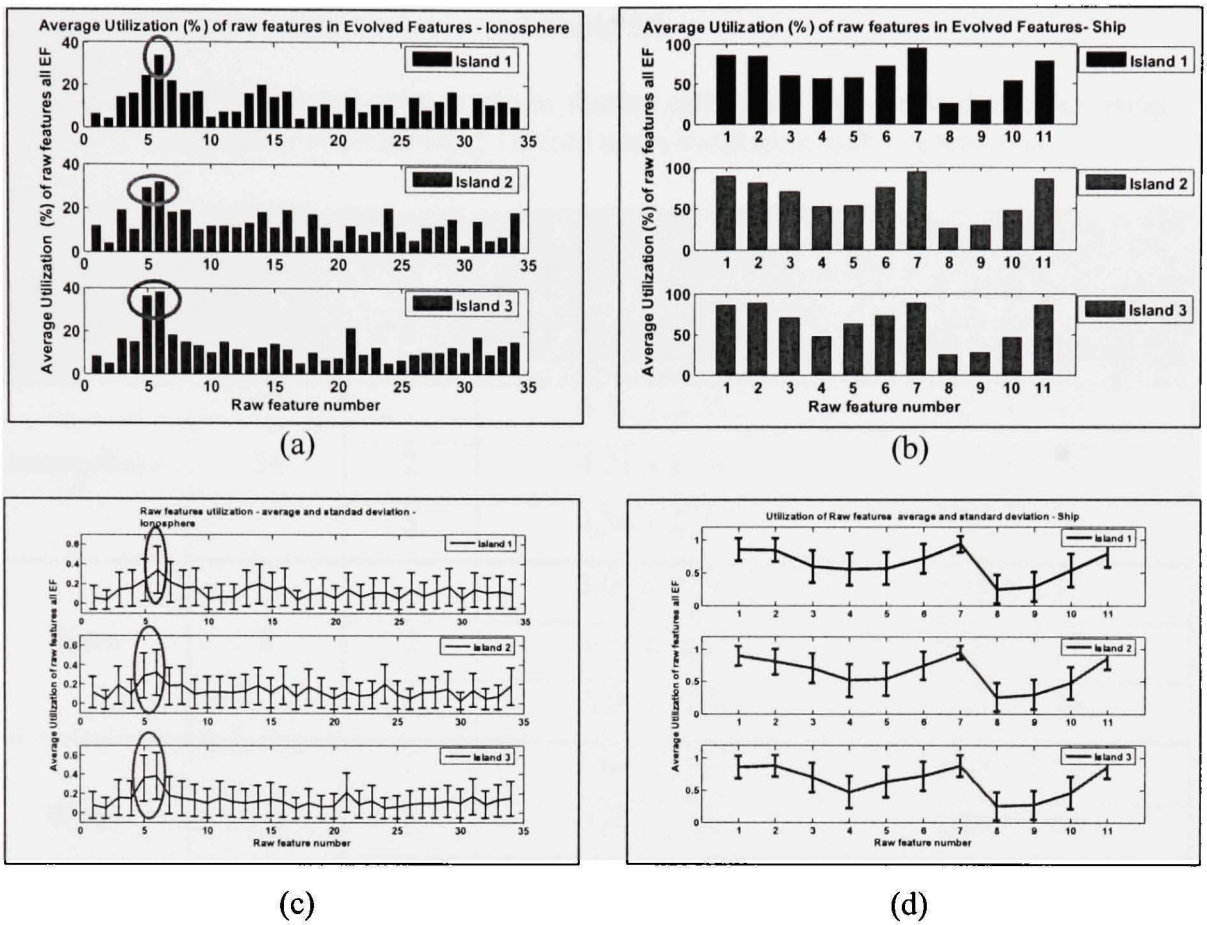


Figure 5.13 ***Raw feature utilization per evolved feature on Ionosphere and Ship data sets.***

To have a deeper view of the situation we have included the average and standard deviation of the utilization of raw features in parts (c) and (d) of Figure 5.13. Again, raw features 4 and 5 have greater influence in Ionosphere (surrounded with ellipses). For the Ship case in part (d), raw features 1, 2, 7 and 10 have the smallest variation around the average value. That could be due to a consistent use of them to build the solutions through the different repetitions and folds.

Figure 5.13 reflects the relative importance of each raw feature in the final solutions (evolved features). We are also interested to know how many raw features, in average, take part in the final solutions.

Table 5.6

Average and standard deviation of raw feature utilization and evolved features using the Island method using 10-fold cross-validation and 5 repetitions

Database	Raw Features	Island	Evolved Features	Raw Features Utilization
			#F*ave* ± Std(#F)	RawF*ave* ± Std(RawF)
Ionosphere	34	1	4.12 ± 2.36	4.21 ± 1.45
		2	4.21 ± 2.28	4.37 ± 1.94
		3	4.36 ± 2.57	4.33 ± 1.90
Pima	8	1	3.09 ± 1.14	5.18 ± 1.17
		2	3.17 ± 1.31	5.04 ± 1.37
		3	3.18 ± 1.20	5.17 ± 1.26
Wine	13	1	3.64 ± 2.15	3.75 ± 1.62
		2	4.20 ± 2.50	3.88 ± 1.66
		3	4.23 ± 2.21	3.72 ± 1.69
Ship	11	1	8.03 ± 1.71	6.97 ± 1.69
		2	8.22 ± 1.49	7.08 ± 1.45
		3	8.23 ± 1.52	6.98± 1.65

Table 5.6 is a comparative between the initial number of raw features, the number of evolved features generated by Bot's method and the number of different raw features in the evolved features. The number of evolved features requires in all cases is a considerable reduction in the representation of the data set. In addition, not all the raw features are required to generate the evolved features. In most cases, less than 50% of the raw features are used. Therefore, the proposed method works at the same as a feature creation and feature selection methodology.

In the case of Ship data set we see that raw features 1, 2, 7 and 11 are important to classify the dataset. This is in agreement to what Valin *et al.*, have reported (Valin et al., 2006).

5.5 General conclusions

Bot's method has been improved by three different strategies: selecting a suitable range of values for genetic operations of cross-over and mutation, transforming the GP algorithm into a coarse-grain version (also called island method) that allows small independent evolutions that interchange some individuals and, finally including a global validation procedure that controls the over-fitting, The theory that supports these three strategies was presented and then some experiments were run to show the results. Some of the conclusions that come out are presented in the following lines.

The adjustment of the GP-related parameters is not an easy task. We have tried instead of using some suitable ranges for some of the main parameters. We have analyzed the influence of population size, number of generations and features (which is particular for Bot's method). We have tested that an initial population of 100 individuals as set in (Bot, 2001) allows a certain degree of diversity within the population. This is shown in figures 5.4 and 5.6 by the red clouds representing the population for different evolving features. The wide range in fitness values (vertical axis) even after having evolved for some features puts in evidence the diversity within the population. Also, individuals with the same fitness values have different size, which also reflects the diversity in the population. The number of generations was fixed to 11 as in (Bot, 2001). Different values, from 11 to 30 were tested and we did not find big differences in the results. The small progress reached with more generations is easily attained with the gain in performance due to a new feature. Thus, short evolutions are one of the advantages of Bot's method.

Different values of cross-over and mutation probabilities were tested and there was no a specific combination that revealed a noticeable better performance overall. We were able to find that a suitable range for cross-over is 0.8 to 0.9 with mutation in the range of 0.05 to 0.1. Therefore, we use in subsequent tests a cross-over of 0.9 and a mutation of 0.08.

Bot's algorithm used a steady state replacement mechanism in which new offspring are immediately introduced in the population and are available for reproduction. In this case, the fitness can not be calculated for all individuals at the same time, so parallelization based on master-slave strategies can not be applied to our algorithm. We have then looked for the island method. We found that it allows reducing the size of the population on each island to 30 individuals and the time computation time is also reduced to less than one third of the time of the standard version. This is not a reduction in resources utilization because we use three islands (or processors) instead one. The real advantage is that the islands explore different zones within the search space and its mechanism generates an improvement in the performance of the final results. In addition, the solution is robust as the evolution can continue even when one of the island stops the evolution (for instance a problem in the processor running the evolution). This is applicable only if the intercommunication towards other island is not completely broken (as would happen in rings topologies with unidirectional migrations). In fact, we presented one of such a case with the Segmentation data set in which, even though one of the evolving islands was stopped for a while, the parallelization method was able to continue and we obtained an increase of the recognition rate in comparison to a sequential version.

In addition to the robust quality of island model of parallelization, we can use the best solution from all islands, which is called the single best individual or single best solution. Instead, we can use all three solutions, V-best solutions (with V=3). The final solution can be calculated as a combination function of the three solutions. A simple combination is based on the majority vote principle presented in Chapter 3. This combination produces improved recognition rates as will be shown in the next chapter.

Bot's procedure generates features that can be used to build classifiers. The evolved features reveal intrinsic relations between the input raw features that are difficult to found out with other searching techniques. The combination of single solutions or classifiers into ensembles is then an alternative to investigate. This is the subject of the following chapter in which we

apply the improved Bot's algorithm to evolve classifiers that will be combined in an ensemble to further increase their performance

The addition of the global validation procedure provides some improvements to Bot's method. It allows the detection of the point in the evolution (feature and generation) where the over-fitting starts and also permits controlling it because we store the individual that best generalizes (testing all the population in a different data set, called validation) and it is used as a final solution instead of the best solution in optimization which over-fits this data set. We have seen that the average number of evolved features using the global validation procedure is increased in comparison to the average calculated with Bot's stopping criterion. Bot's stopping criterion is made at the end of the evolution for every feature and in practice, works as an early stopping because it discards the evolved feature that do no represent a considerable gain in optimization.

Besides, global validation procedure shows if the number of generations and features set *a priori* are suitable for our problems. If the point where global validation indicates to stop (when over-fitting starts) the evolution is too different in comparison to the number of features used, we can adjust it accordingly. For instance, in the Pima data set, we see that the average number of evolved features can be set to a maximum of 5. There is no need to await the evolution for 10 features, because it reveals that no improvement is reached further than 5 features. In the other hand, global validation (and the island method) revealed that Wine and Segmentation data sets can be evolved until 10 features and certain level of gain is attained. So, we can modify an initial setting of 5 evolving features to 10. In general, we can better adjust the stopping point to obtain the best performance without over-fitting too much the optimization data set.

Global validation showed that, in some cases, over-fitting can appear at any point in the evolution (see figures 5.8 and 5.9) and it affects subsequent evolving features, because they depend on all previous evolved features in Bot's method. Therefore, subsequent solutions have an implicit over-fitting that is difficult to reduce. Once detected this implicit over-

fitting, there is no mechanism to avoid it because the best solution in optimization can not be changed by the best solution in validation. This would be an alteration to the evolution process. The global validation procedure must remain a mechanism that keeps an eye on the evolution without interfering with it.

CHAPTER 6

FEATURE CREATION FOR ENSEMBLES

We have mentioned before that the main elements that take part in the design of EoC are: data, classifiers and the combination functions. Data samples can be split between different classifiers or the attributes (features) can be divided in different sub-sets. Classifiers used in the ensemble can be all of the same type or mixed types. Different combination functions can be used to generate the ensemble response (Dietterich, 2000). Our ensemble design methodology splits the data attributes into different subsets using the RS method, generates base classifiers as a set of related evolved features (or formulas) using GP as method and combines the generated classifiers by majority vote. The RS method was presented in Chapter 3 and the generation of classifiers using Bot's method and GP was explained in detail in Chapters 4 and 5.

In the standard RS method, each base classifier has a partial view of the data i.e. a subset of the features available in the training data. This method relies on a stochastic process that randomly selects a number of features from the given set to construct each classifier (Ho, 1998a). The classifier is trained with the feature sub-set randomly selected. In our case, the classifier will be generated by constructing evolved features from the RS applied to it. The stochastic selected features are the input to Bot's method that generates a set of evolved features (or a set of formulas) as the classifier. So, the generated classifier discovers intrinsic relations between the input features and creates a set of formulas that best represents the input RS. The new representation only utilizes the most important raw features from the input RS, which implies an additional reduction of the number of raw features used. In brief, Bot's method unveils important features and exploits its inter-relation to optimize the base classifier. Then, the optimized base classifiers are to be combined using majority vote to generate the ensemble.

This methodology will be applied to a practical problem to analyse the benefits of generating base classifiers via GP towards the construction of ensembles.

The first part of this chapter describes the generation and selection of ensembles of *k*-NN classifiers with the RS method in the context of isolated handwritten digit recognition, as developed in (Tremblay, 2004). Selecting the most performing and diverse classifiers generates optimized ensembles. Section 6.2 presents the re-engineering process applied to the base classifiers based on Bot's method, focused on ensemble generation. The experimental protocol followed and the results are presented in Section 6.3.

6.1 Optimization of ensembles of *k*-NN by random subspaces

This section describes the optimization of ensemble of *k*-NN classifiers using the RS method in the context of a problem of isolated handwritten digit recognition, as developed in (Tremblay, 2004). We have mentioned before that the main elements that take part in the design of EoC are: data, classifiers and the combination functions. The data comes from the data set NIST SD19, which Oliveira represented as a vector of dimension 132 (Oliveira *et al.*, 2002). This will be described in detail in Section 6.3. The classifier used was a *k*-NN and the combination function is majority vote. In a first set of experiments Tremblay determined the number of base classifiers, the number of neighbours taken in each *k*-NN classifier, and the cardinality of the subspaces to have the best performance (Tremblay, 2004). It was found that ensembles of 100 *k*-NN with k=1 and a cardinality of 32 features produce the best results. In addition, Tremblay found that the ensemble gives the best performance when *k*-NN classifiers were trained with data sets of 5000 samples or less (Tremblay, 2004). The comparison of performance was made against a single *k*-NN and an optimized multilayer perceptron (MLP). The specific input features for each *k*-NN classifier were randomly selected from the original set of 132 features without replacement.

Once the 32 features for each of the 100 *k*-NN are chosen, the 100 base classifiers are set. Afterwards, Tremblay applied different searching methods to find out the best ensembles.

Each base classifier is associated to a specific set of 32 features, so searching methods look for minimum number of base classifiers that maximize the ensemble performance. Two ensembles with the same number of base classifiers could have different performance because the particular *k*-NN classifiers included in each ensemble are built with different subspaces. The structure of each ensemble is defined by the amount of base classifier and the index (from 1 to 100) of the *k*-NN classifiers included. This information is the outcome from the searching method managed by a GA (single or multi-objective) guided by the global performance of the ensemble as objective function. In the case of a multi-objective GA, another additional searching criterion, like diversity, is used in combination with performance. Each individual in the GA population is a vector of 100 elements, i.e. each component reflects the index of the classifier included (when the bit is 1) or not (bit in zero). Population size is 128, which means the number of candidate ensembles evaluated in each generation. In brief, GA have been used to select ensembles of *k*-NN classifiers from a pool of 100 base *k*-NN classifiers in such a way that performance (of the ensembles created) is maximized. Figure 6.1 represents the base classifiers, the GA-population and the selection of ensembles. According to Tremblay, as the evolution progresses, more individuals are similar or even equal to the best one (on optimization). In the end, only a few individuals are different i.e. only a few ensembles are different (Tremblay, 2004). This experiment is replicated 30 times and the best ensemble is selected for each replication (Tremblay, 2004).

The searching methods that give best performance results in (Tremblay, 2004) are presented in Table 6.1. The first three lines of Table 6.1 show the references values used to compare the ensemble results: a single *k*-NN with 132 features, the ensemble of 100 *k*-NN and a MLP network with the whole set of 132 features as input. Since each experiment is repeated 30 times, the three last lines of Table 6.1 show the best result for each case as reported in (Tremblay, 2004).

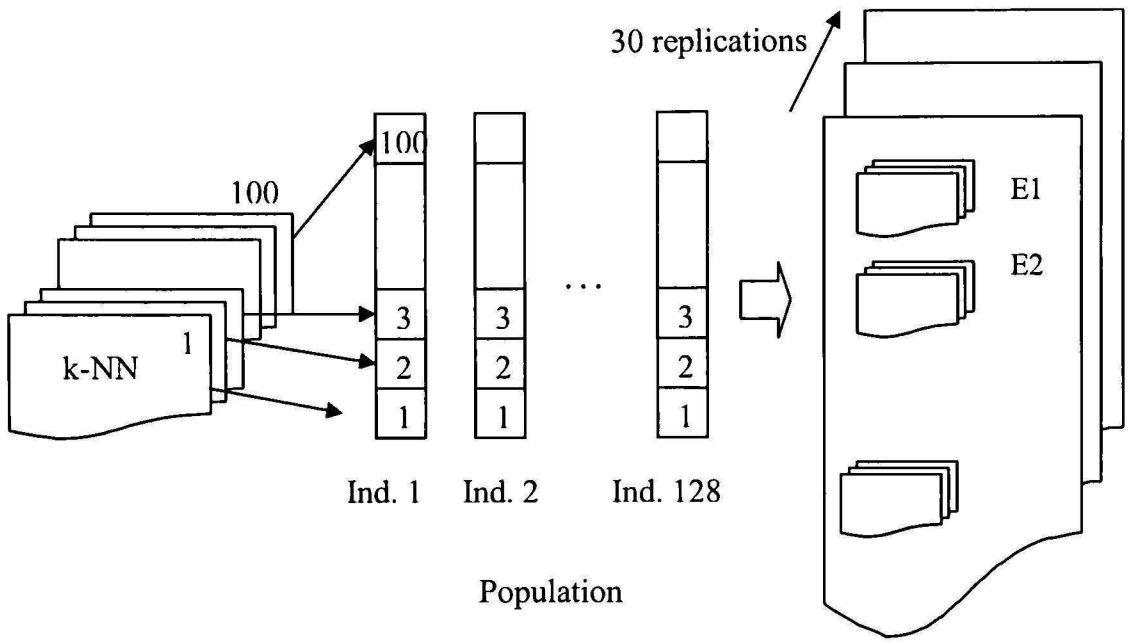


Figure 6.1 ***Original ensemble, GA-based population and selection of ensembles.***

Table 6.1

Results from optimization of ensembles with GA. Extracted from (Tremblay, 2004)

Method	# Classifiers	Rec. rate on OPT	Rec. rate on VAL	Rec. rate on TEST
k-NN (132 feat.)	1	93.23%	93.33%	93.34%
Ens. 100 k-NN	100	95.79%	96.09%	**96.28%**
MLP (132 feat.)	1	96.11%	95.91%	95.27%
Ranking	76	95.93%	96.15%	**96.26%**
Simple GA	31	96.82%	96.43%	**96.41%**
NSGA-e	24	96.84%	96.29%	**96.40%**

From the reference values, we see that an ensemble of 100 k-NN with only 32 features each one gives higher recognition rate than a single k-NN using the whole set of 132 features. The best reference value comes from the MLP network that uses all 132 input features. From the

results we can summarize that the optimized ensembles provide a better recognition rate with smaller ensembles (less base classifiers).The ensembles generated by ranking reduce in approximately 25% the number of base classifiers and maintain the recognition rate compared to the rate of the ensemble of 100 *k*-NN classifiers. Results from single GA and a multi-objective GA (in this case Non-dominated sorting GA NSGA) that jointly minimizes the error rate (which is 1 – Rec. Rate) and the number of classifiers generate a reduction between 70 to 75% in the number of classifiers and, in addition, an increase in recognition rate. This time, the performance is better than the MLP network. Table 6.1 presents the results in three different data sets: Optimization, Validation and Test. In our experimental protocol, we use the same data sets (see Section 6.3).

We can conclude that the creation of ensembles with the RS method generates smaller and more performing ensembles. Therefore, the exclusion of redundant features produced with RS method helps to increase the recognition rate and reduces the number of required base classifiers. We formulate the question: Is redundancy between raw features still present in each subspace? If so, can it be reduced? Does this reduction generate an increase in recognition rate?

We will apply the improved Bot's method to optimize the already built classifiers with the RS method. The optimization will show if the resulting classifiers produce a boost in performance of the ensemble and (or) a reduction in the cardinality. This is the subject of the following sections of this chapter.

6.2 Creation of ensembles of classifiers based on GP

The creation of ensembles of GP-based classifiers splits attribute data (also called raw features) into different sub-sets using the RS method, generates base classifiers as a set of related evolved features (or formulas) using GP and combines the generated classifiers, called hereafter evolved classifiers, by majority vote. Each one of the components is described in the following paragraphs.

Previous section explained the RS method applied to a specific problem of isolated handwritten digit recognition using the NIST SD19. Oliveira *et al.*, (Oliveira *et al.*, 2002) created a description of the digit images as vectors of 132 features. This description has been used in different publications in the fields of pattern recognition and machine learning with encouraging results (Oliveira *et al.*, 2002; Oliveira *et al.*, 2003a; Dos Santos *et al.*, 2006; Ko *et al.*, 2007; Radtke *et al.*, 2006; Tremblay, 2004). The RS splits the initial set of attributes in sets of 32 randomly selected raw features. A detailed description of the data set used is presented in the experimental protocol.

In (Tremblay, 2004, Tremblay *et al.*, 2004) each ensemble is composed of a series of *k*-NN classifiers generated by RS method RS_i, $i=1,...,R$, where R is the number of classifiers in the ensemble. The feature construction process for ensembles takes the *i-th* classifier RS_i, $i=1,...,R$, as input to the improved Bot's method. Each subset of features is different and depends on the selection made by the RS method. The process of transforming a RS into the evolved classifier is called re-engineering of base classifiers and it is explained in the following section.

6.2.1 Re-engineering of base classifiers

Figure 6.2 shows a detail of the input to the GP algorithm. Through the evolution process, GP generates evolved individuals, which are equations or formulas in terms of its input features (specific subset of raw features from each classifier). GP evolution seeks for individuals that generate a high recognition rate, by measuring the fitness of each individual in the population. Fitness measure is associated to the performance achieved by the evolved solutions on the optimization data set for each feature and each generation. Fitness measure of each individual in the population is calculated by a *k*-NN or a MDM classifier. As mentioned in Section 5.1.2, we will not use fitness measured based on Fisher criterion.

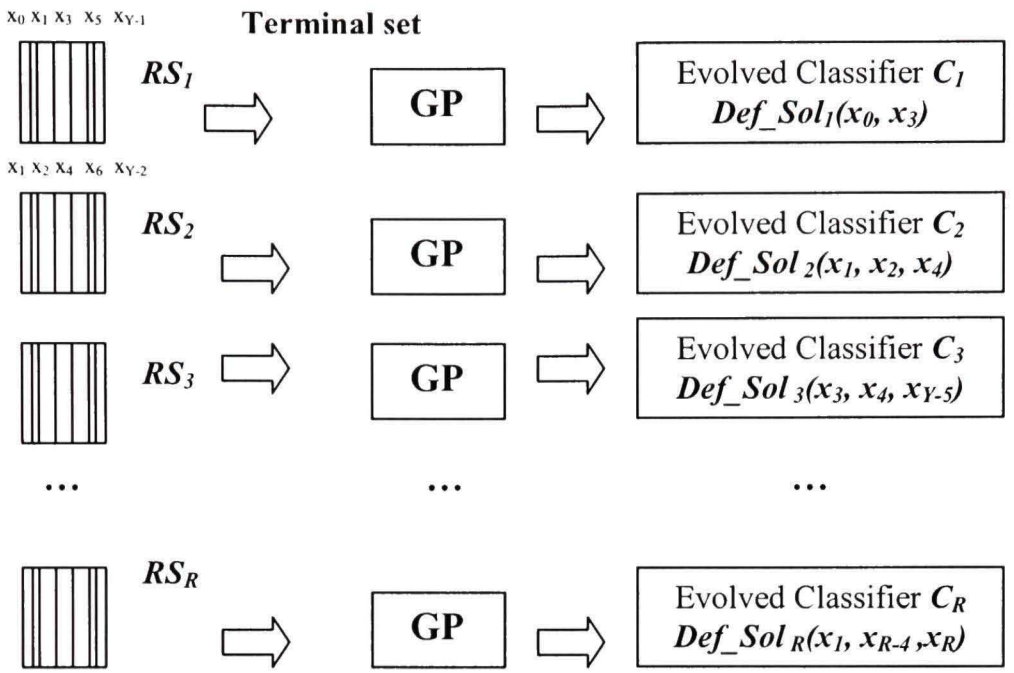
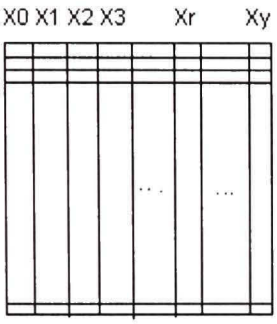


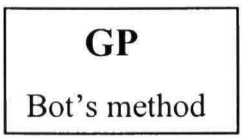
Figure 6.2 ***Input set of features (generated by random subspaces) to a GP algorithm.***

The combination of the classifier and the GP algorithm is called here the GP-classifier wrapper. Figure 6.3 shows a typical example of evolved features created by a GP run according to Bot's method: the evolved classifier *Ci* ($1 \leq i \leq R$) can be represented, in this case, as a composition of three features $Def\_Sol_i = \mathrm{EF}_{1i} \oplus EF_{2i} \oplus EF_{3i}$. The base classifiers are generated applying Bot's method which uses GP to discover tacit relation between the input variables and makes them explicit in the functions that it generates. A full description of Bot's method was presented in Chapters 3 and 4. The input to the algorithm and outputs from it, are depicted in Figure 6.3 below. The input is constructed as follows (Ho, 1998a): given a set of examples in an N-dimensional space (N=132), N_{RS} -dimensional subspaces are considered (N_{RS} =32)

$$\left\{\left(x_1, x_2, x_3, ..., x_N\right) \mid x_j = 1, j \in I, x_j = 0, j \notin I\right\}, \tag{6.1}$$

where I is an N_{RS}-element subset of $\{1,2,3...,N\}$, and $N_{RS} < N$.





Evolved Classifier C

$$C = Def\_Sol(x_0, x_1, x_2, x_3, ..., x_Y) = EF_1 \oplus EF_2 \oplus EF_3$$

where,

$$EF_1(x_0, x_3, x_5, ..., x_{Y-3}) = \frac{(x_0 + x_3^2) * x_{Y-3}}{x_{25} * x_7 * 0.789};$$

$$EF_2(x_1, x_2, x_4, ..., x_{Y-2}) = (\frac{x_8 + x_{22}}{x_{10}}) * \log(x_{20})$$

$$EF_3(x_2, x_5, x_{10}, ..., x_{30}) = (\frac{x_{25} - \sin(x_{15})}{0.5647}) + x_7$$

Figure 6.3 ***Creation of evolved features using GP and Bot's method.***

Each evolved classifier $\boldsymbol{C_i}$ is used to classify its corresponding input generated by the random subspace RS_i. The advantage of this process is that the evolved function finds relations between the input features in such a way that this new classifier $\boldsymbol{C_i}$ has a recognition rate similar or better than the original classifier RS_i. To analyze the effectiveness of the re-engineering procedure, we compare and analyse the recognition rates of RS_i and $\boldsymbol{C_i}$ (see Figure 6.4, above). Each random subspace RS_i has associated a recognition rate of the *k*-NN classifier when the 32 features are used as input to it. This recognition rate RR(RS_i) is a single number. In the other hand, the recognition rate of the corresponding evolved classifier $\boldsymbol{C_i}$, noted here as RR($\boldsymbol{C_i}$), is a collection of recognition rate values according to the number of evolved features used (for example in Figure 6.3, RR($\boldsymbol{C_i}$) will be a set of three recognition rate values: RR(EF_1), RR($EF_1 \oplus EFf_2$) and RR($EF_1 \oplus EF_2 \oplus EF_3$).

6.2.2 Global Validation on evolved classifiers

We suppose that the ensemble is composed of R classifiers (as shown in Figure 6.2). The process described in Section 6.2.1 is done on each of them. For each random subspace RS_i considered, the solution found in optimization $Def\_Sol=EF_1 \oplus EF_2 \oplus EF_3$ represents the evolved classifier $\boldsymbol{C}_i$. Each of the R evolved classifiers $\boldsymbol{C}_i$, has been created by using the optimization data set projected on the 32 input features managed by its corresponding RS_i, by means of the formula contained in the evolved classifier $\boldsymbol{C}_i$. As mentioned in Section 4.2, during GP evolution (optimization), solutions can learn so much the data presented that they may over-fit it. Therefore, we have to establish mechanisms to control the over-fitting.

At a first level, no validation procedure is used (detailed explanation in Section 5.3.1). In this case, the best individual found on the last generation of optimization process is directly used to calculate the recognition rate on the test set. When each of the R evolved classifiers $\boldsymbol{C}_i$ is projected over the test set, the performance without any validation is obtained for each evolved classifier $\boldsymbol{C}_i$ for $i=1,...,R$. Each of the R solutions may over-fit its corresponding optimization data. To identify when the over-fitting is produced and to control it, we implemented the global validation procedure.

As indicated in Section 5.3.3, global validation requires that for each generation and each feature, the individuals (evolved functions) of the population are projected onto the validation data set and then the auxiliary archive is updated with the individual with best performance on validation. This process is run at the same time that the optimization process. Global validation procedure was described in Section 5.1.3.

In the end, two evolved individuals are retained: the best individual generated in optimization and the best individual obtained with the global validation procedure. The latter procedure identifies the best individual in validation and the auxiliary archive contains information about the time (which means the feature and generation numbers) when this individual was included in the auxiliary archive S. From this moment, no further improvement is achieved,

so the evolution can be stopped at this point preventing the evolution from over-fit the optimization data set. If the best individual in the optimization is compared to the best individual in validation, we detect the occurrence of over-fitting. Comparing the numbers of feature and generation in the auxiliary file against the number of features used in the evolution we can identify if the stopping criterion used in optimization was placed at the right point or, it has already over-fitted optimization data set. See additional details in Section 5.3.6. The best individual found in optimization (over-fitted solution) and the best individual found with the global validation process are used to calculate the performance of the evolved classifier C_i on the test set and perceive the difference when using global validation.

The analysis to make in this case is to compare the recognition rates of each evolved classifier C_i without validation process (evolved classifier C_i obtained with optimization and projected on the test set) and the recognition rate of each evolved classifier C_i obtained with global validation. This could reveal the ranges in over-fit. See Figure 6.4 above.

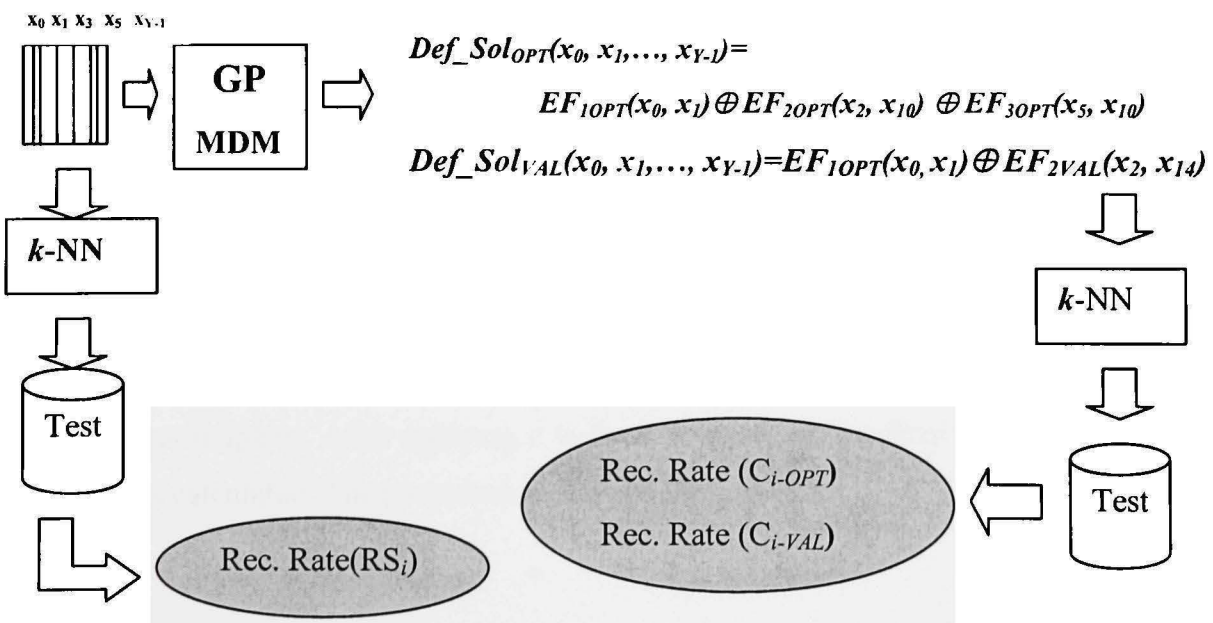


Figure 6.4 ***Analysis of results of re-engineering process.***

Comparison of recognition rates of input RSi against evolved classifier without global validation $C_{i\text{-}OPT}$ and evolved classifier with global validation $C_{i\text{-}VAL}$

In conclusion, it is hoped that each new classifier $\boldsymbol{C_i}$ would be a function of a reduced set of the 32 raw input features of the input classifier RS_i with a recognition rate similar $RR(\boldsymbol{C_i})$ or sometimes better than $RR(RS_i)$.

As mentioned below, the evolution process of a classifier RS_i for $i=1,\ldots, R$ is repeated for all R classifiers considered. It is important to note that although the process to evolve each of the R classifiers RS_i, $i=1,\ldots,R$ is the same, the input feature set used during the evolution of the GP algorithm is different (a different terminal set as shown in Figure 6.2).

6.2.3 Combination of re-engineered base classifiers

Once the evolution process is done and the global validation procedure has been applied, the set of re-engineered base classifiers can be combined to produce the ensemble. As mentioned in Section 3.3, there are different combination functions. We use simple majority vote. A variant, called weighted majority vote is also used. The weight is defined as the recognition rate of each evolved the classifier that takes part in the ensemble.

Initially, the ensemble is composed of R classifiers that have evolved independently. Each evolved classifier created by the global validation procedure $C_{i\text{-}VAL}$ for $i=1,\ldots,R$ is applied to the test data set. In order to decide the class of each sample of this data set, every $C_{i\text{-}VAL}$ gives its vote and the ensemble decision is taken by majority vote. It means that the class which receives the most of the votes is assigned to the tested sample. Ties are randomly decided for one of the participants. After applying it to the whole test set, the final recognition rate of the ensemble is calculated. The procedure is shown in Figure 6.5.

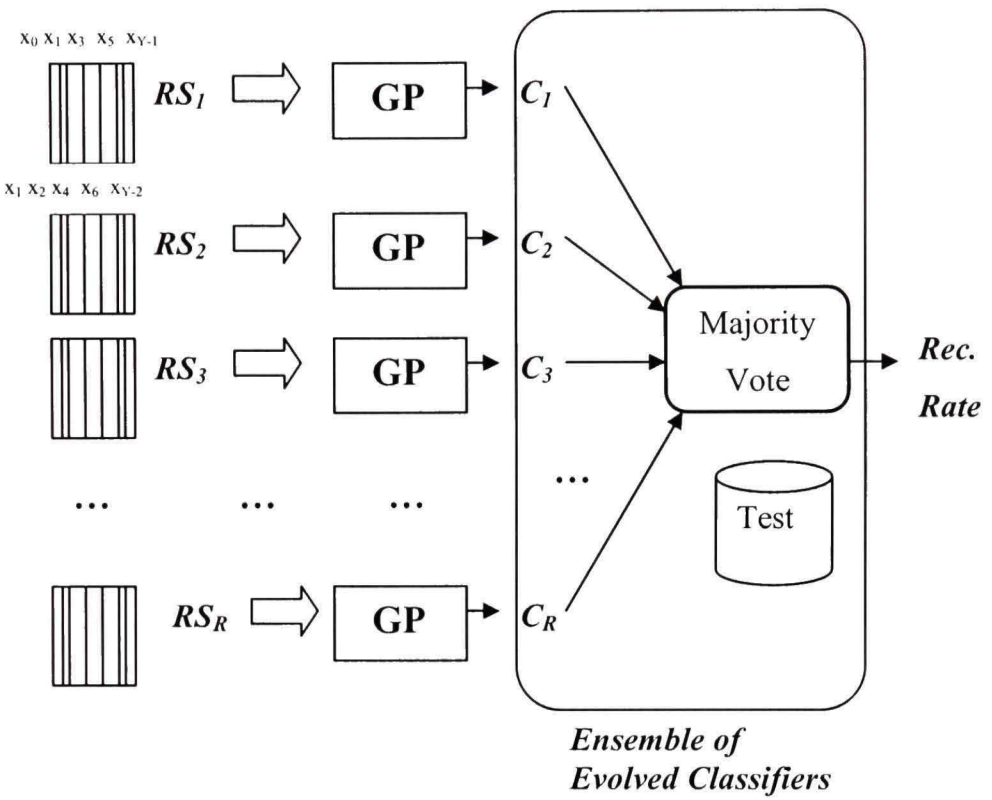


Figure 6.5 ***Feature creation for ensembles.***

Random subspaces RSi are inputs to Bot's algorithm, based on GP, which produces evolved classifiers C_i (also called re-engineered classifiers). Each classifier has been built independently. They are put together to generate the ensemble, using a combination by majority vote on the test set.

The whole process can be summarized as follows: initially an ensemble of *k*-NN classifiers is built using the RS method. This method is applied in the context of a problem of isolated handwritten digit recognition and it was developed in (Tremblay, 2004). The ensemble is optimized by searching the best ensembles (with fewer base classifiers and recognition rates similar or higher than the original ensemble of 100 *k*-NN classifiers) via GA. We take one of these ensembles and applied a re-engineering process to each base classifier, noted as RS_i for $i=1,...,R$. The re-engineering process consists in applying Bot's procedure individually to each input RS_i. Bot's method is based on GP and it creates a set of evolved features as function of the raw features composing each RS_i. The evolved features are noted as C_i, $i=1,...,R$. By means of the global validation procedure incorporated to Bot's method, the

evolved feature controls the over-fitting phenomenon. Afterwards, the evolved classifiers $C_{i\text{-}VAL}$, $i=1,...,R$ found with the global validation procedure are applied to the test data set. Decision of the ensemble is taken by majority vote.

6.3 Experimental protocol – Feature creation for ensembles

This section describes the experiments developed to generate ensembles of classifiers. Base classifiers are represented as evolved features generated by Bot's method that uses GP as evolutionary mechanism. To test our ensemble creation method, we have applied it to the problem of isolated handwritten digit recognition, using the data set NIST SD19.

In the first subsection we briefly present the NIST SD19 data set, its description as a vector of 132 raw characteristics and how the data is partitioned into the four disjoint blocks for optimization, training, validation and test. Then, the preparations to be done before evolving each base classifier are detailed in the second subsection. In the third subsection, we make an overview of the tests to run. Finally, the results are presented and analyzed in the fourth sub-section.

6.3.1 NIST SD19 database description and usage

Feature creation for ensembles is used for an application of isolated handwritten digits classification. Images of isolated handwritten digits come from the data set NIST-SD19. Digits from 0 to 9 are considered to produce a pattern recognition classification problem of 10 classes. Different features are extracted from the original images to produce a representation as a vector. Each element of the vector codes the value of a feature extracted from the image. Following sub-sections explains how the vector of features is created and the use of NIST-SD19 segments to conform the data sets required for creating GP-based features for ensembles.

6.3.1.1 Feature set

The initial description of an isolated handwritten digit is made by a fixed size vector of 132 components, proposed by Oliveira *et al.* (Oliveira *et al.*, 2002). Vector components include concavity and contour measurements. Digit image is divided into 6 zones as shown in Figure 6.6(a) (taken from (Oliveira *et al.*, 2003a)). Concavity measurements intend to find what surrounds each pixel in the image and contour measurements extract information about slant and number of points of borders. Concavity measurements provide 78 features as follows: the image is divided in 6 zones and each one is characterized by 13 features, then 6x13=78. Each set of 13 features contains information about number of black pixels in specific directions or a combination. Directions used are: 4-Freeman and four auxiliary directions s_1, s_2, s_3 and s_4 as shown in Figure 6.6(b) taken from (Oliveira *et al.*, 2003a).

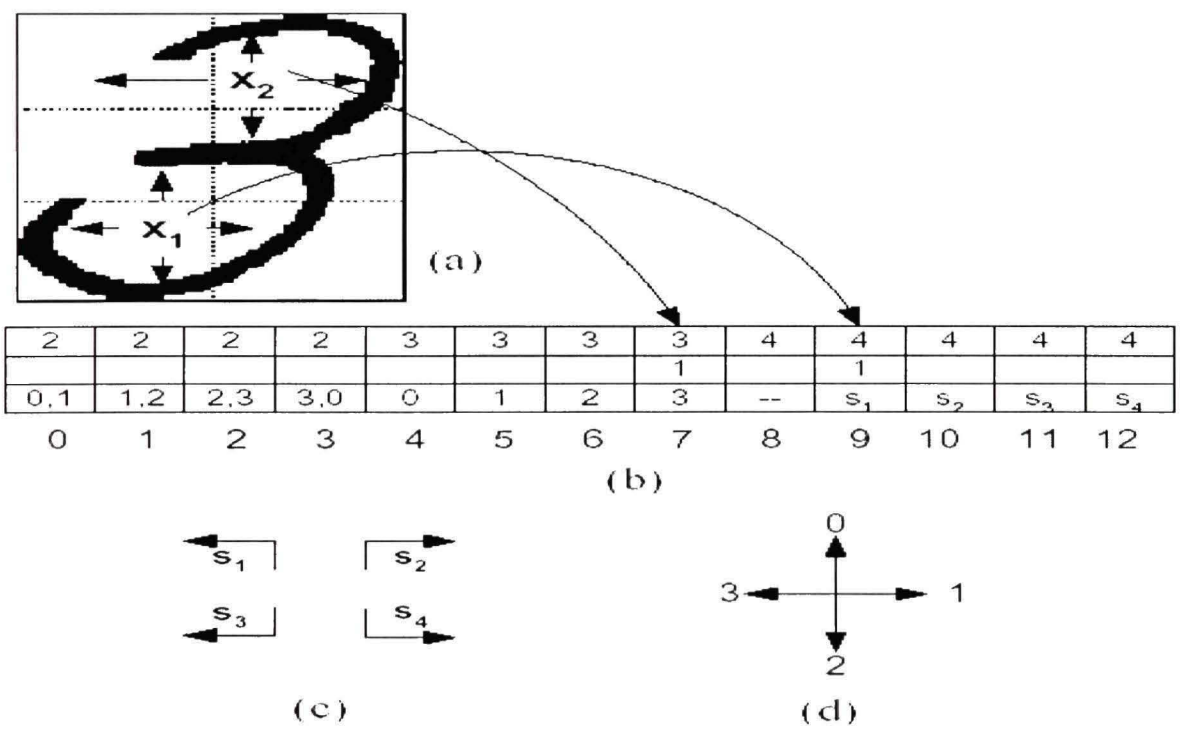


Figure 6.6 ***Concavity measures for NIST-SD19 samples (Oliveira et al., 2003a).***

Concavity measurements: (a) Concavities, (b) feature vector, (c) auxiliary directions, and (d) 4-Freeman directions

Contour measurements extract information about slant and number of points of borders. Figure 6.7(a) presents contour from digit 5. One of the 6 zones is expanded in this figure taken from (Oliveira *et al.*, 2003a). Histogram of each of the 8-Freeman directions is recorded in a vector as shown in Figure 6.7(b). As a result, there are 48 features (8 directions and 6 zones). Values are normalized in a range 0 to 1. In addition, another feature counts the number of pixels in each zone and is normalized between 0 and 1. This part generates 6 new features.

In total a digit image is initially represented by 132 features:

78 concavity features	+	**54** concavity features	=	**132** features
6 zones * 13 feat each zone	+	(8 direct. * 6 z.) + 6 feat		

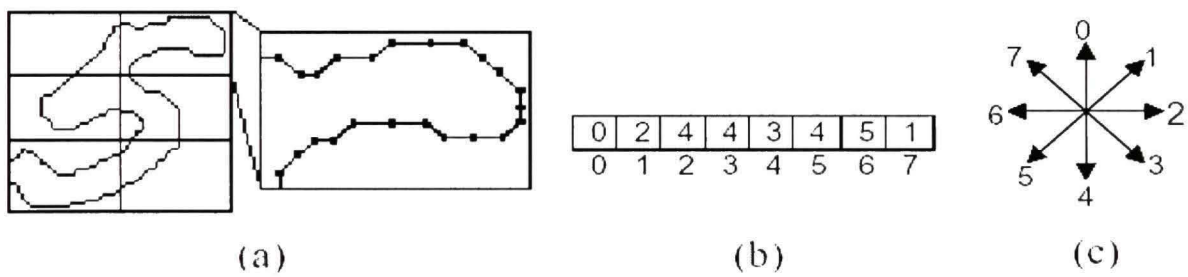


Figure 6.7 ***Contour measures for NIST-SD19 samples (Oliveira et al., 2003a).***

Contour measurements: (a) Contour image and zone expanded, (b) feature vector, (c) 8-Freeman directions

Figure 6.8 presents an example of images of isolated handwritten digits extracted from NIST-SD19.

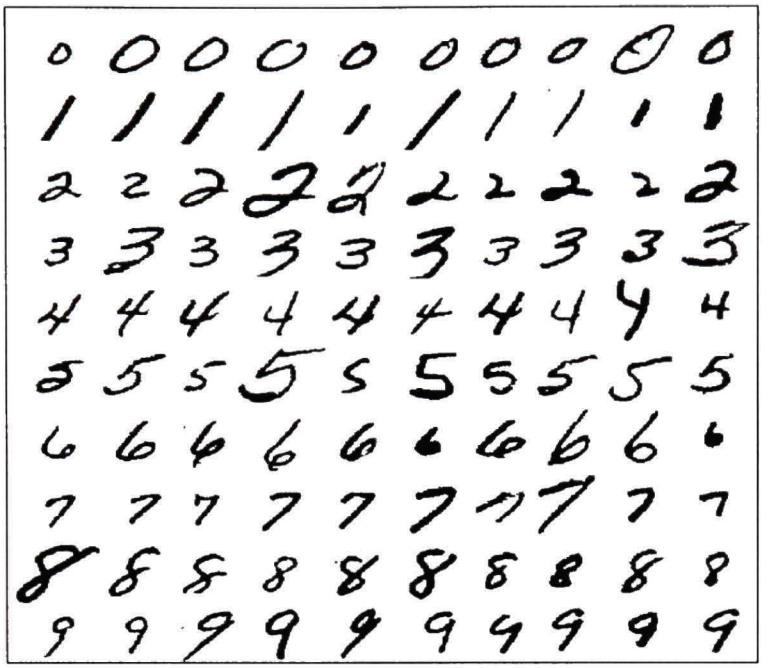


Figure 6.8 ***Example of digit images from NIST-SD19 database***

A vector of 132 features is extracted from each pre-processed image to build the initial representation.

6.3.1.2 Database description and usage

NIST SD-19 database contains images of digits from 0 to 9, extracted from eight handwritten sample form (hsf) series. It was originally divided in three sets: hsf-{0123}, hsf-7 and hsf-4. The first set hsf-{0123} contains 195000 examples, hsf-7 contains 60089 examples and hsf-4 contains 58646 examples.

To evolve each base classifier we use the projection of the data over a specific RS as input, in the same way as defined in (Tremblay, 2004). So, we show the partitions as used in (Tremblay, 2004) and therefore we can compare the results. According to (Tremblay, 2004), the best EoC was chosen by means of a stochastic search guided by a single or multi-objective GA with performance as the objective function (for multi objective GA an additional objective is used, for instance cardinality or a measure of diversity). The training phase in this case, used 5000 examples from hsf-{0123}. During the optimization process, different ensembles were obtained. This phase was carried out with 10000 examples coming

from hsf-{0123}. To choose the best ensemble, a validation data set of 10000 examples also coming from hsf-{0123} was used. Finally, performance of the best ensemble selected was measured with a test dataset. For this phase, datasets hsf-7 and hsf-4 were used. It is worth to mention that the subsets from hsf-{0123} used for training, optimization and validation are disjoint one to the others.

We use the same partition of NIST-SD19data as in (Tremblay, 2004), which is presented in Table 6.2. Training of *k*-NN and GP algorithm will use the first 5000 examples from hsf-{0123}. Optimization process to find the best evolved individual for each classifier of the best ensemble will be implemented by using the same 10000 examples coming from hsf-{0123} that were used during optimization of the selection of ensembles: examples 50001-60000. Global validation strategy will be deployed by using the same 10000 examples from hsf-{0123} that were used during validation of the selection of ensembles (examples 60001-70000). Finally, test of the performance of the created features for the selected ensemble will use datasets hsf-7. In this way performance obtained with the feature construction can be compared to values obtained in the selection process.

Table 6.2

Different data sets to be used during feature construction for ensembles

Partition of NIST-SD19		
Description	**Quantity**	**Range**
Training dataset: hsf-{0123}	5,000	hsf-{0123}:1-5000
Optimization dataset: hsf-{0123}	10,000	hsf-{0123}:50001-60000
Validation dataset: hsf-{0123}	10,000	hsf-{0123}:60001-70000
Test data set 1: hsf-7	60,089	hsf-7: 1- 60,089

6.3.2 Preparations before evolving a base classifier

The ensemble creation technique used evolves each base classifier independently and combines them using majority vote as a fusion mechanism. The procedure of evolving each base classifier was referred as re-engineering of base classifiers and it consists in generating successive evolved features according to Bot's method which uses a GP algorithm as evolutionary technique.

In the experimental protocol in Section 4.4, the fitness measure of individuals in the population was measured in two different ways: recognition rate of a classifier (k-NN or MDM) or the inter-class scatter over the intra-class scatter using the Fischer criterion. Results showed that performance values are very close practically in all cases. Even though, there is a considerable difference in the computation time required to calculate the fitness with a k-NN and a MDM classifier. Therefore, we choose a MDM classifier to be used during the evolutionary searching phase and qualify the evolved features with both classifiers (k-NN and MDM) when testing the selected (validated) solutions. It is worth to mention that solutions when mapped onto the validation set are also measured with MDM classifier. As commented in Section 4.4.2.1, MDM classifier does not require any parameter to be set in advance. For the k-NN classifier case, we use k=1 as default value and increase it to 11 to resolve ties in the testing phase.

Results from Section 5.1 show that setting optimal GP parameters is a monumental task and we could not find an optimal cross-over and mutation probabilities pair applicable to four different data sets from UCI repository tested. Different works agree on this and emphasize that it is more practical to set GP parameters within feasible ranges (Fernandez *et al.*, 2003; Koza, 1992). We have used a cross-over probability of 0.9 and individual mutation probability of 0.08. This combination showed to generate results that are better than the average in our analysis of four different data sets from UCI repository (see Section 5.1.1). Table 6.3 reviews the settings used for the GP runs. Other GP parameter values (see Table 4.4) have been set to Open BEAGLE default values.

Evolution of classifiers is carried out for 32 features, and 11 generations (10 generations plus the initial random generation). The maximum number of features *Max_Nb_features* was established by looking the performance of the resulting evolved classifier. We tested this parameter for 5, 10, 16 and 32 features. Performance for 5 and 10 features was poorer than the original RS. Performance for 16 features was similar to the original RS only in a few cases and was lower in the other cases. At 32 features, the recognition rate of evolved classifiers was consistently similar or higher than this of the original RS. We also tested 48 features and its performance was not so different than in the case of 32 features. In contrast, the additional computation time made it not practical. An evolving classifier is built by splitting the population between three islands that evolve at the same time and interchange some individuals as described in Section 5.2. Each of the three islands has a population of 30 individuals and 2 individuals migrate every two generations to all other islands (in total 4 migrants go to other islands and it receives 4 individuals from the neighbouring islands). Migrant individuals are randomly selected and individuals replaced in the receiving island are also randomly selected. This prevents from exercising a big selection pressure that could speed up the evolution at expense of a possible premature convergence as suggested in (Whitley *et al.,* 1999; Skolicki and De Jong, 2005).

In the function set, the functions sinus, cosines, logarithm and exponential have been added. These unary functions require only one terminal element as input and can be useful to reduce the size of the individuals. They have already used in the experiments in Chapter 5 (see Figure 5.4).

Table 6.3

GP related parameters for evolving classifiers

Objective	To find out the minimal number of mathematical expressions that maximize recognition rate on optimization data set	
Terminal set	32 variables within the set { $(x_1,\ldots,x_q,\ldots,x_{132})$, ephemeral random constant}, depending on the raw features included in each base classifier created by RS. Variables are normalized according to equation (4.3). Ephemeral random constants are random floating point constants ranging from -1.0 to 1.0	
Function set	{+, -, *, /, exp, log, sin, cos}. Division (/) refers to protected division to avoid division by zero	
Fitness measure	A classifier that maps the last evolved mathematical expression in conjunction with previous evolved mathematical expressions into any of the dataset classes by using the optimization data set. Penalty according to tree size decreases the fitness measure.	
	Global Validation: Evolved individuals are mapped into the validation set for each feature and each generation. Penalty also used.	
Raw fitness	The number of optimization samples correctly classified	
Parameters	Population size	90 in total (30 each island)
	Number of generations	11
	Selection	Tournament selection 2 individuals
	Cross-over probability (individual)	0.9
	Mutation probability (individual)	0.08
	Creation Initial population	Ramped half-and-half
	Maximum tree depth	17
	Replacement mechanism	Steady –state
	Elitism	Keep one individual
Success predicate	Trees' Individual representation with classification rate equal to 99.99% (over validation data set). Success does not stop evolution.	

6.3.3 General description of experiments to run

The ensemble feature creation method is tested from two different perspectives. In the first one, we choose base classifiers with different performance and combine them to create the ensemble. The criterion in this case is focused on base classifiers with very different performance rates: from the worst to the best and then building the ensemble. The analysis of results in this case will indicate whether the feature creation for ensembles method is useful

or not: the performance of the ensemble is better than its components even though we use base classifiers with poor performance. In the second approach, we use the base classifiers that compose an EoC selected by a GA (single or multi-objective) used to maximize the recognition rate of the ensemble (in the single GA case) or to minimize at the same time, the error rate and the cardinality (multi-objective GA). In this case, the reference is the ensemble performance and not their component performance. Results from the latter approach will show if the ensembles created based on evolved classifiers bring advantages compared to ensembles built with *k*-NN classifiers applied to raw RS.

6.3.3.1 Selection of base classifiers with different performances

From an initial pool of 100 base classifiers we selected 9 based on the recognition rate in the optimization data set. The nine selected classifiers are used to build the ensemble. Initially we explain how the classifiers were selected and then we explain how the ensemble was built. Figure 6.9 shows the diagram of the selection process of base classifiers.

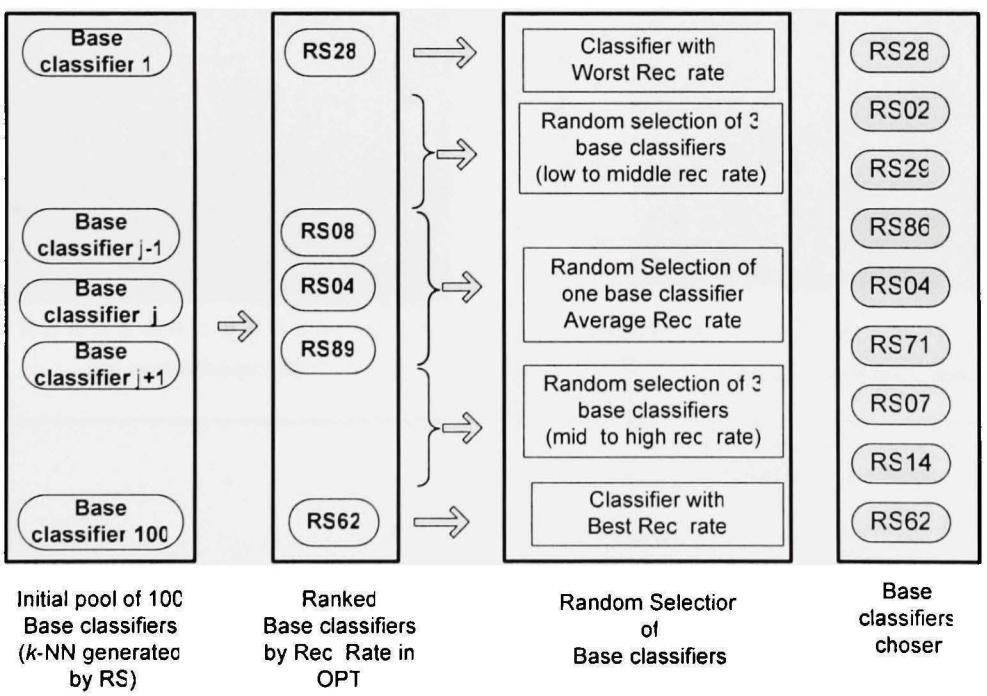


Figure 6.9 ***Diagram of selection of base classifiers with different performances.***

In order to ensure that the selected base classifiers have recognition rates that truly represent the whole initial pool the process is not fully random. The selection process has three steps (see Figure 6.9). In the first step, the classifiers in the pool are ranked according to the recognition rate in the optimization data set. From the ranked set, we select the worst and the best base classifiers as part of the second step. They correspond to RS28 and RS62 with recognition rates of 82.9% and 91.49% respectively. Then we select one base classifier with a recognition rate close to the median value. To do that we choose 5 base classifiers around the median and we randomly select one from these five classifiers. And the third step is as follows: from the base classifiers in the range from low to middle recognition rates, we randomly selected three. In the same way we randomly selected other three classifiers from the range of middle to high recognition rates. Each range has been indicated by a curly bracket in Figure 6.9. In this way we ensure that the selected base classifiers have really different recognition rates that represent the whole range.

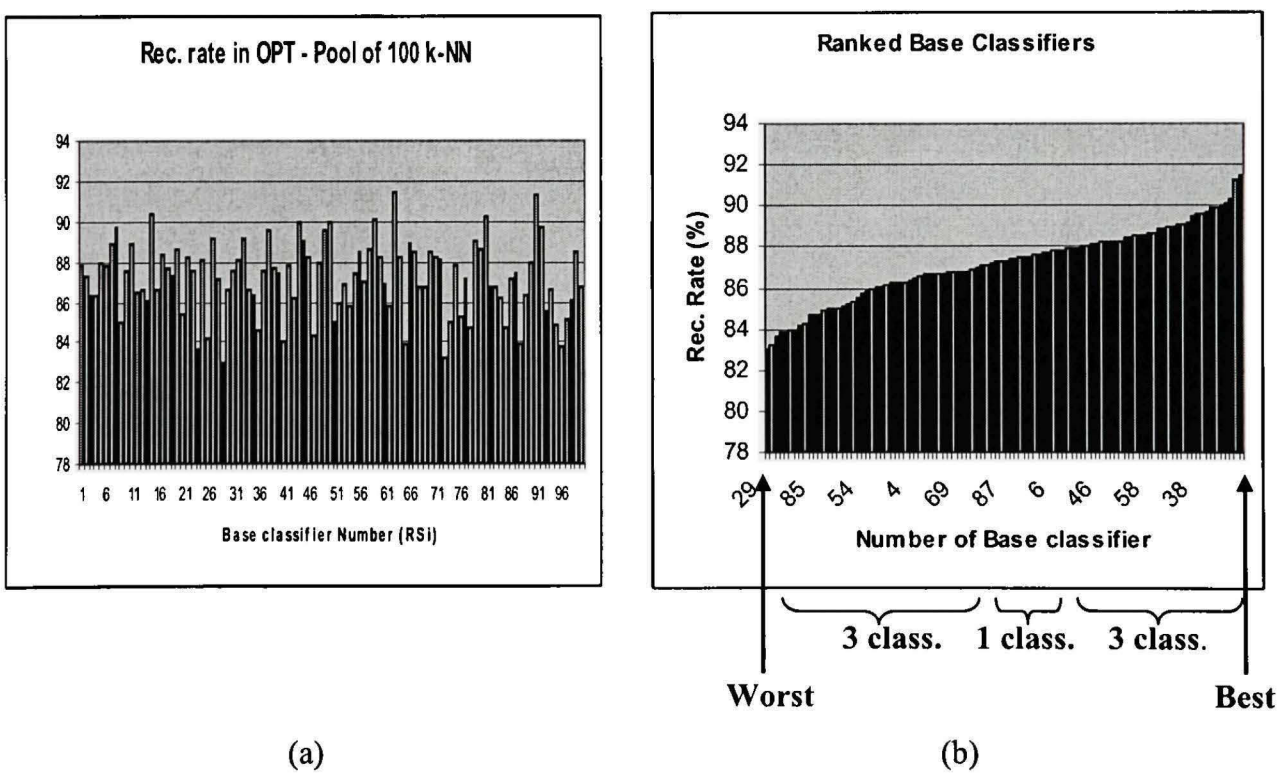


Figure 6.10 ***Ranges of selection of base classifiers with different recognition rates.***

Figure 6.10(a) shows the initial pool of base classifiers with the recognition rate and Figure 6.10(b) indicates approximately the ranges of selection of base classifiers and also the base classifiers with worst and best recognition rate. The selected base classifiers from worst to best in the optimization data set are: RS28, RS02, RS29, RS86, RS04, RS71, RS07, RS14 and RS62 (see Figure 6.9).

Each base classifier RS_i for $i=1,\ldots, R$, with $R=9$, is evolved by splitting the population into three islands, using Bot's method for 32 features. Evolved classifiers C_i are combined feature by feature using majority vote as fusion function. As the island method generates solutions for each island, we can build an intermediate ensemble with the three islands of each evolved classifier, named Ensemble of Island Classifier i and noted as $EoIC_i$. Thereafter, we combine the R $EoIC_i$ for $i=1,\ldots, R$ to generate the final ensemble of evolved classifiers EoC. In the other hand, we can combine altogether the $3*R$ evolved classifiers to generate the final EoC. In the last case, each evolved island is considered as a component of the final ensemble. Section 6.3.4 presents the results for those two cases and the analysis.

The following experiments have been run: Nine base classifiers have been evolved based on Bot's algorithm using the Island method. An intermediate ensemble of island classifiers is created as solution for each base classifier. In total, 9 intermediate ensembles are created ($EoIC_i$, with for $i=1,\ldots, R=9$). Thereafter, the final ensemble is generated (EoC) with nine component ensembles. In the second approach, evolution of base classifiers based on island method creates 9*3=27 evolved classifiers that are combined to generate the final ensemble. In both cases, decision of the final ensemble is taken by majority vote feature by feature. It means that the components of the ensemble vote to take the decision about the first evolved feature of the ensemble, then about the second evolved feature and so on until the last evolved feature (32). As based in Bot's algorithm, recognition rate of the ensemble grows as new features are included in the ensemble. Thus allows us to establish a number of required evolved features to attain a recognition rate level.

6.3.3.2 Selection of an ensemble of base classifiers

In this case, we start with an already built ensemble and we rebuild it, this time with evolved classifiers. Tremblay used single GA and NSGA-II to search the best sub-ensemble from a pool of 100 *k*-NN created with RS (Tremblay, 2004, Dos Santos *et al.,* 2006). The sub-ensembles built with single GA, have an average of 40 classifiers (Tremblay, 2004). As mentioned before, our process of evolving base classifiers is very consuming in time and resources, so we look for an ensemble of acceptable recognition rate but with a minimum number of base classifiers. When the searching algorithm NSGA-II uses recognition rate and a diversity measure (coincident) as objectives, the cardinality of some of the resulting ensembles is about 7 or less, which is appropriate for our purposes. Consequently, we decide to use ensembles with these searching criteria. The complete procedure is as follows: from an initial pool of 100 *k*-NN created with RS, NSGA-II searches for sub-ensembles with recognition rate and coincident diversity measure as objectives and we take the best set of solutions in validation (Pareto front). That gives us a set of ensembles. Then, a selection of a single ensemble is run in three phases as presented in Figure 6.11. In the first phase, we take the 14 booked ensembles with cardinality less than 5 classifiers: we obtain 14 different ensembles. In the second selection phase, we analyze the frequency of occurrence of the classifiers within the ensembles. Four classifiers (RS96, RS90, RS72 and RS43) appeared in 11 of the booked ensembles. Finally, in the third phase, we randomly select one ensemble from the subset of 11 ensembles that contain at least three of the most frequently used classifiers. Figure 6.11 shows a diagram of this procedure and the outcomes at each stage. The final selected ensemble was composed by RS43, RS91, RS95, RS96 and RS90. This ensemble has a recognition rate of 94.68% in optimization and 95.06% in test.

In order to build the evolved ensemble we have to evolve each base classifier and then combine them using majority vote. In the previous section, we built the ensemble combining the votes feature by feature. Since over-fitting can appear at different points through the evolution, the ensemble of evolved classifiers combined feature by feature could suffer from over-fitting even using the global validation procedure.

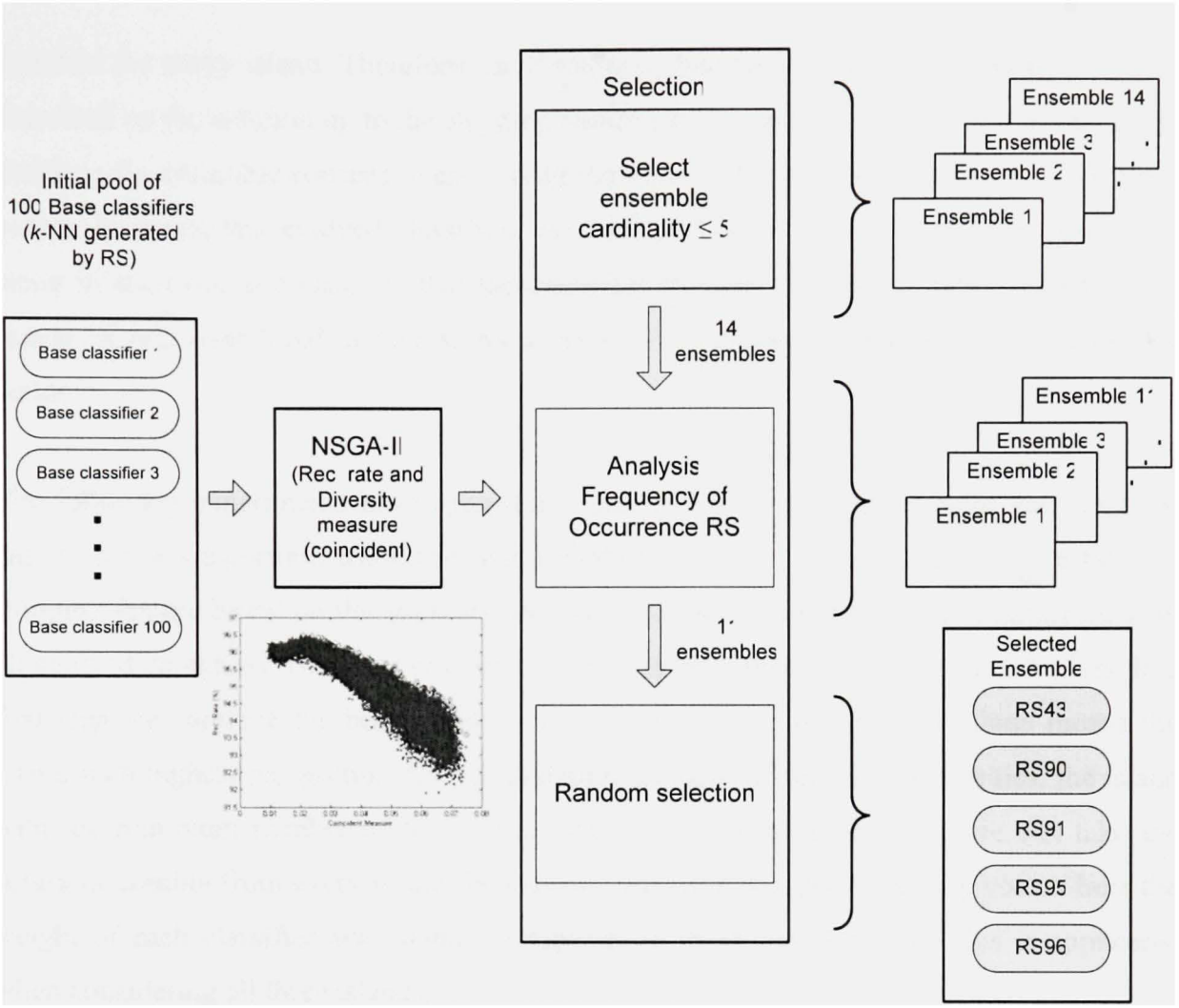


Figure 6.11 ***Diagram of selection of already defined ensembles.***

To avoid this potential situation, we use a different approach in this case. For each of the evolving classifiers that compose the ensemble, we look for the last feature where solutions do not over-fit the optimization data. Therefore, the number of features of each evolved classifier can be different. For instance, suppose that the ensemble is composed of two classifiers (C_1 and C_2), C_1 uses 17 features and C_2 uses 23 features. The resulting ensemble will combine the solutions and votes at the corresponding number of features. The number of evolved features to use by each classifier is automatically discovered by means of the auxiliary archive generated along the global validation procedure and they can be identified as the earliest feature at which no further improvement in performance on the validation data set is produced. Since evolution is based on island method, the mentioned stopping point is

searched for every island. Therefore, each evolved classifier to use in the ensemble will be described by the solution up to the stopping feature and the corresponding votes at this point. Building the ensemble consists in combining the votes of the evolved classifiers as described before. It means that evolved classifiers can have different number of features, taking the better of each one and ensuring that the ensemble (combination of evolved classifiers) is a fusion of non over-fitted elements. As a result, the ensemble solutions would generalize better.

The following experiments have been run: each of the base classifier have been evolved based on Bot's algorithm using the island method. For each evolved classifier we find the stopping feature based on the auxiliary archive. The final ensemble (EoC) is generated with all evolved classifiers. Different criterion can be used to combine the evolved classifiers. In a first step we combine the best island from each evolved classifier. Best Island means the island with highest recognition rate in validation. In cases of recognition rate ties, the island with the minimum number of features is then chosen. In a second step, we can take the solutions coming from every island. In addition, we test a weighted majority vote, where the weight of each classifier and island corresponds to its recognition rate. This is applicable when considering all three islands.

Results are compared to the recognition rates of the ensembles built when using k-NN classifiers generated with all features from RS. Following section presents the results of the experiments mentioned here with the analysis of results.

6.3.4 Construction of ensembles with base classifiers of different performances

6.3.4.1 Methodology

The base classifiers were originally built (Tremblay, 2004) as k-NN (with k=1) that use as data the projection of the data set onto the specific RS. So, we note the original base classifiers as RS28 to simplify the notation. From the worst to the best recognition rate in

optimization the base classifiers used were: RS28, RS02, RS29, RS86, RS04, RS71, RS07, RS14 and RS62. Each one was evolved with Bot's algorithm and applying the island model with the parameters indicated in Section 6.3.2 and Table 6.3 and Table 4.4 (additional parameters). The set of three islands create an intermediate ensemble $EoIC_i$, where index i=1,..., R refers to the number of evolved classifiers that will compose the final ensemble. Combination of all three islands, by majority vote on test set, produces a recognition rate curve as displayed in Figure 6.12 which presents the diagram of the intermediate ensemble generation and then the final decision of the ensemble.

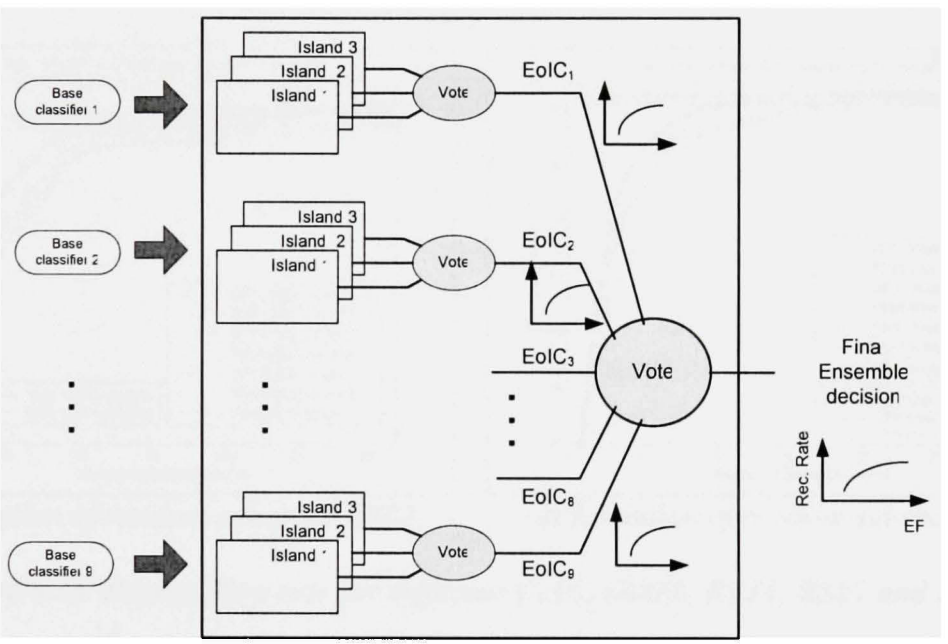
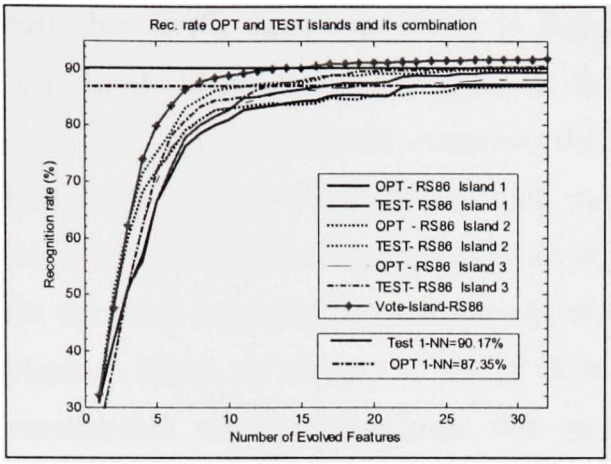


Figure 6.12 ***Diagram of Intermediate Ensemble and final ensemble.***

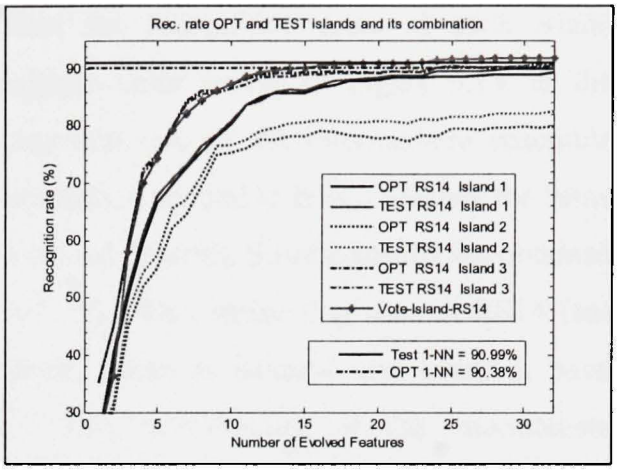
Intermediate ensembles EoICi are built with the vote of each island for every feature. Then each EoICi's vote is combined feature by feature to generate the final ensemble decision

6.3.4.2 Creation of evolved classifiers EoIC

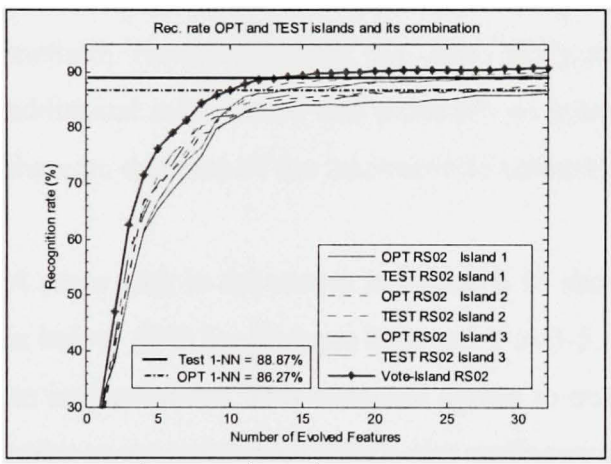
Figure 6.13 shows how the recognition rate in optimization and test varies as the number of evolved features increases, for each island. In addition, it shows the recognition rate of the raw RS with 1-NN classifier as two horizontal lines.



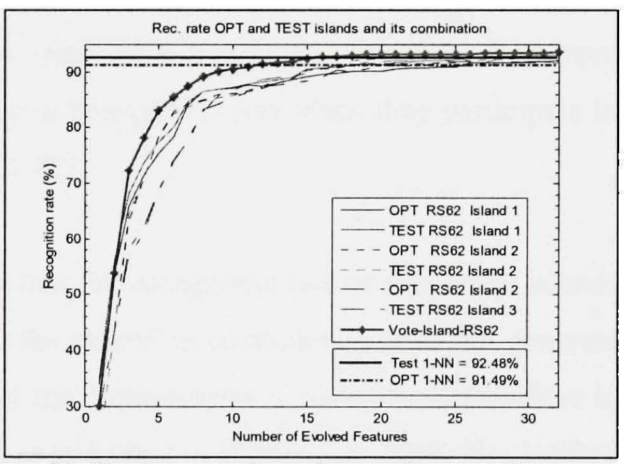
(a) Evolution of random subspace RS86.



(b)Evolution of random subspace RS14.



(c) Evolution of random subspace RS02.



(d) Evolution of random subspace RS62.

Figure 6.13 ***Recognition rate for different EoIC$_i$ (RS86, RS14, RS02 and RS62).***

Recognition rate for every island and for the ensemble of islands by majority vote (test set). Recognition rate on the optimization (in blue) and on the test (in red) sets for each island of the system. Results from intermediate ensemble of islands EoICi shown in magenta color. Results are compared to recognition rate on optimization and test when using raw random subspaces and 1-NN

The solid line represents the recognition rate in test and the dash-dotted line represents in optimization. Combination of all three islands, by majority vote on test set, produces a recognition rate curve displayed in magenta color. Recognition rate attained at each feature is indicated by a start marker also in magenta color. In particular, Figure 6.13(a) shows the results of the evolved RS86, (b) shows results for RS14, (c) shows results for RS02 and (d) shows results for RS62. It can be seen that the resultant recognition rate of the intermediate

ensemble EoIC*i* (magenta color) is higher than the recognition rates of each island considered independently. As shown in the magenta color curves of Figure 6.13, as the number of evolved features increases, the recognition rate of the intermediate ensemble EoIC*i* also improves. In Figure 6.13 (a), the intermediate ensemble $EoIC_4$ reaches the same recognition rate level of the raw RS at about 15 evolved features. Similar results are obtained for the other intermediate ensembles (EoIC*i*, *i=1,..,9*). One interesting case is RS14 (see Figure 6.13(b)), in which two out of three islands, noted as island-1 and island-2, have considerable worse performance than island-3. But, performance of the intermediate ensemble EoIC*i* is better than island-3 alone. This can be consequence of the exploration of different regions of the searching space by each island and the diversity created by the island method. Notwithstanding the low recognition rates of island-1 and island-2, they have additional information that produces an increase in recognition rate when they participate in the vote decision of the intermediate ensemble EoIC*i*.

A close look to the curves in Figure 6.13 shows that the recognition rate of individual islands is below 50% for features between 1 to 3-5, so the ensemble combination does not generate an improvement. This outcome agrees to one of the requirements of individual classifiers in order to get ensembles with better performance, as indicated in (Dietterich, 2000; Ho, 1998a)

Table 6.4 presents the results of the evolved RS for each island ensemble. We include the recognition rate values when the solutions selected in validation are evaluated with the test data set, the average number of evolved features in these solutions and the recognition rate obtained when using raw RS with 1-NN classifiers. Since the evolution is made for 32 features, we take the evolved features as the number where the recognition is maximal in the validation set. The results in the test set are similar in the two cases. The improvement in our strategy can be due to the use of the island ensemble. The results in validation and test for each island are shown in in Annex VI.

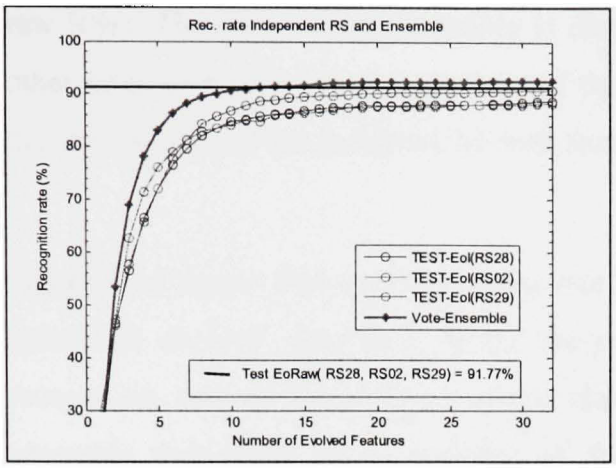
Table 6.4

Comparison of recognition rate and standard deviation on test set for raw RS with *1*-NN classifiers and evolved classifiers

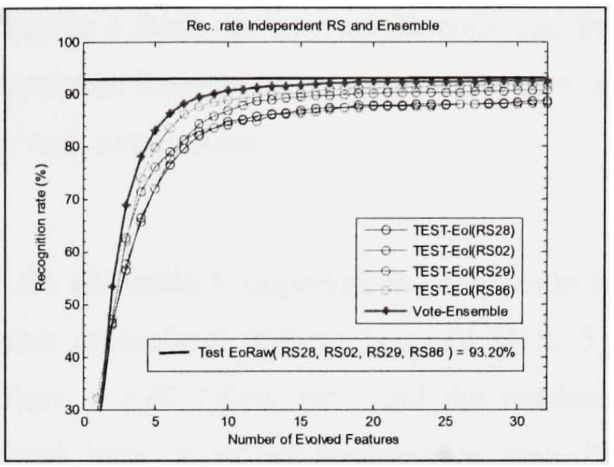
RS	Rec. Rate (%) Evolved classifiers TEST	#*Fave*	Rec. Rate (%) TEST Raw RS with 1-NN
RS02	88.08±1.29	23.2±1.29	88.87
RS04	91.61±1.85	22.9±2.85	89.28
RS07	92.50±1.62	23.2±2.14	91.40
RS14	91.00±2.25	20.4±3.53	90.99
RS28	88.08±1.29	23.2±1.29	85.71
RS29	87.68±2.15	21.2±2.15	87.85
RS62	93.28±1.89	21.7±2.85	92.48
RS71	90.67±1.53	21.8±2.88	89.88
RS86	90.64±1.40	24.5±3.53	90.17

Once all the classifiers have been evolved, they can be combined to generate the ensemble. In this opportunity, we have created intermediate ensembles of islands EoIC*i* where i=1,..., R, with R=9. The nine intermediate ensembles are generated from the random subspaces RS28, RS02, RS29, RS86, RS04, RS71, RS07, RS14 and RS62. Intermediate ensembles EoIC*i* will be added one by one, starting from the worst $EoIC_1$ generated from RS28 until the best intermediate ensemble $EoIC_9$ generated from RS62. All nine will compose the final ensemble of evolved classifiers. Final ensembles are also created feature by feature.

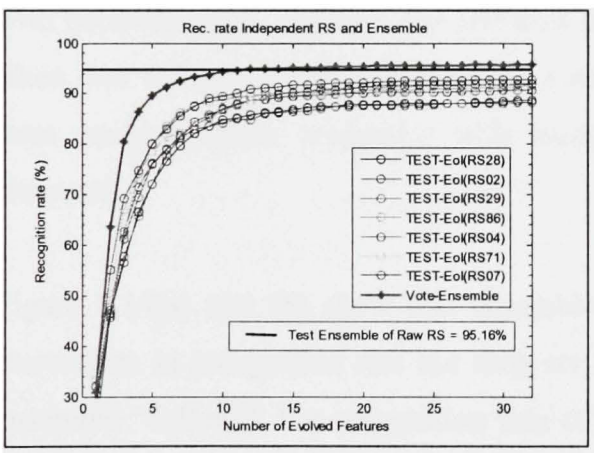
Figure 6.14 shows different steps in the creation of the final ensemble: (a) with three intermediate ensembles, (b) with four, (c) with seven and (d) with all nine intermediate ensembles.



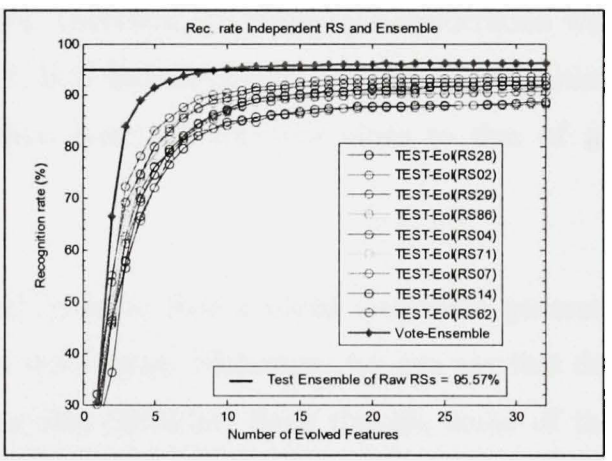
(a) Ensemble generated with three intermediate ensembles from RS28, RS02 and RS29. Recognition rate of the same ensemble using 1-NN is 91.77%, meanwhile the maximum obtained here is 90.507% with 20 evolved features.



(b) Ensemble generated with four intermediate ensembles from RS28, RS02, RS29 and RS86. Recognition rate of the same ensemble using 1-NN is 93.20%, meanwhile the maximum obtained here is 92.27% with 22 evolved features.



(c) Ensemble generated with seven intermediate ensembles. Recognition rate of the same ensemble using 1-NN is 95.16% meanwhile the maximum obtained here is 95.55% with 21 evolved features.



(d) Ensemble generated with all nine intermediate ensembles. Recognition rate of the same ensemble using 1-NN is 95.57% meanwhile the maximum obtained here is 96.14% with only 21 evolved features.

Figure 6.14 ***Recognition rate for ensemble of evolved classifiers***

Curves in Figure 6.14 show the recognition rate of some of the ensembles created. Each figure includes the recognition rate of the elements of the ensemble and the recognition rate of the generated ensemble and the recognition rate of the reference ensemble (created with

raw RSs). The curve of the ensemble is displayed as a thick line in magenta color and the other lines correspond to the elements of the ensemble. Recognition rates are calculated on the test dataset and are indicated for each feature with star markers.

Figure 6.14 shows that the recognition rate of the ensemble is improved in comparison to individual evolved classifiers. When the number of evolved features is small (1 to 3), recognition rate of individual evolved classifiers is well below 50% and the resultant ensemble (for these small number of features) have a worse performance than its components. As recognition rates of individual classifiers increase along with the number of features, the corresponding performance of the ensemble improves and, in some cases, in a considerable amount. It is worth to mention that RS selected were taken to have different level of performance in optimization, from the worst to the best and taking as well some RS with performance in between the previous limits. Therefore, no diversity consideration was taken into account in this selection. As a result, EoC that have high individual recognition rates could generate ensembles with recognition rate that are very close to that of its elements.

Figure 6.14(a) and (b) show that ensembles of three or four evolved classifiers generate increments in recognition rate but they are not quite large. Moreover, we can say that the ensemble "follows" the recognition rate of the best classifier. Even though, some of the classifiers of the ensemble have a very low recognition rate, performance of the ensemble is not deteriorated with respect to the best classifier. Some degree of diversity between different RS could explain this result. In the other hand, when intermediate ensembles with high performance are part of the ensemble, recognition rates are significantly increased as shown in

Figure 6.14(c) and (d). Parts (c) and (d), show that performance of the ensemble for features greater than 10 does not augment appreciably. But the most appealing point is that ensembles achieve a high performance with a few evolved features: it is enough to use 10-12 evolved features.

Table 6.5

Recognition rate on Test set for different EoC(1-NN) and average recognition rate and standard deviation for ensembles of evolved classifiers, including cardinality

Ensembles and their base classifiers	*EoC(1-NN)*		GP-evolved classifiers		
	Rec. Rate (%) TEST	Card.	Rec. Rate (%) TEST	Evolved Features	Card.
RS28, RS02	86.72	2	87.64±1.35	21±2.08	6
RS28, RS02, RS29	91.77	3	*90.50*±1.52	22±1.89	9
RS28, RS02, RS29, RS86	93.20	4	92.27±1.86	20±3.12	12
RS28, RS02, RS29, RS86, RS04	94.39	5	94.52±1.47	22±2.47	15
RS28, RS02, RS29, RS86, RS04, RS71	94.54	6	94.88±1.36	23±2.53	18
RS28, RS02, RS29, RS86, RS04, RS71, RS07	95.16	7	95.55±1.54	21±2.08	21
RS28, RS02, RS29, RS86, RS04, RS71, RS07, RS14	95.28	8	95.86±1.66	28±2.85	24
RS28, RS02, RS29, RS86, RS04, RS71, RS07, RS14 and RS62	95.57	9	96.14±1.29	21±2.17	27

Table 6.5 reviews our results and compares them to ensembles created with raw RS and 1-NN. Small ensembles with three or four elements and 1-NN classifiers perform better than ensembles of evolved classifiers. As the number of elements of the ensemble increases, the evolved classifiers produce ensembles with higher performance. The ensembles shown in Table 6.5 increases one by one the cardinality when the elements are raw RS and increases by three in the case of ensembles of evolved RS due to the intermediate island ensemble. Differences in performance are not significant, but the real advantage is the number of evolved features required to reach this performance level that are in the range of 19 to 23. In case of ensembles of 1-NN classifiers, the whole set of 32 raw features is used. In the following section there is an analysis of the of raw feature utilization.

6.3.5 Construction of ensembles with already selected base classifiers

6.3.5.1 Methodology

In this case, we start with an already selected ensemble of raw RS classifiers and re-build the ensemble but this time based on evolved classifiers. As explained in Section 6.3.3.2, the smallest ensemble found is composed by the following classifiers: RS43, RS91, RS95, RS96 and RS90. This ensemble has a recognition rate of 94.68% in optimization and 95.06% in test.

In this subsection we describe different approaches to build ensemble of re-engineered classifiers. Initially we present the recognition rates and number of features used for each evolved classifier (and island in it), so we can compare them to the recognition rates of raw RS and *1*-NN classifiers. Building the ensemble consists in combining the votes of the evolved classifiers with different number of features, in such a way that the ensemble is a fusion of non over-fitted elements. In the following section we show how to identify the number of evolved features for each classifier (and island) based on global validation.

Votes from evolved classifiers can be combined in different ways to build the ensembles. In a first step, we combine the best island from each evolved base classifier. The term best island means the island with highest recognition rate in validation. In cases of recognition rate ties, the island with the minimum number of features is then chosen. In a second step, we engender an intermediate or island ensemble. This intermediate ensemble is built with three evolved classifiers combined with majority vote. The three evolved classifiers correspond to the solutions from the three islands when evolving each RS. In a third phase, we take all the solutions coming from every island and base classifier and combine all of them to build the ensemble. The combination function can be simple majority vote (SMV) or weighted majority vote (WMV), where the weight of each classifier and island corresponds to its recognition rate. WMV is applicable when considering all the islands.

6.3.5.2 Evolved Classifiers Ci

Each base classifier was evolved with Bot's algorithm and the island model with the parameters indicated in Section 6.3.2 and in Table 6.3 and Table 4.4. The dispersion of the recognition rates for each island of the evolved classifiers is presented in the upper part of Figure 6.15.

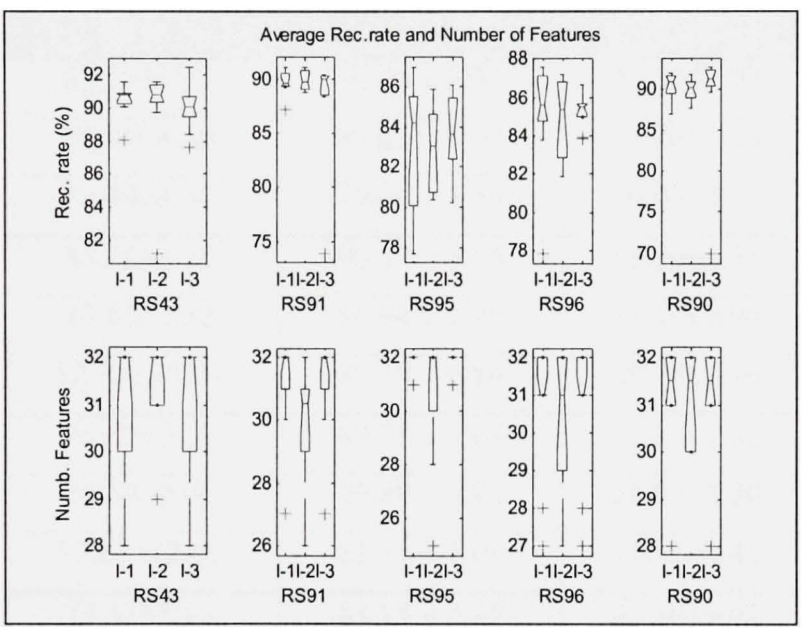


Figure 6.15 ***Recognition rate and number of evolved features for each evolved classifier.***

The lower part of the figure shows the dispersion of the number of evolved features for each classifier and island used. Recognition rate is calculated over the test set. It can be seen that results are very similar between islands of the same evolved classifier. The number of evolved features required is close to the 32 raw features. Table 6.6 details the results by showing the recognition rate of each evolved classifier (and its islands) in validation and test.

Table 6.6

Average recognition rate and standard deviation in Validation and Test for evolved RS, including number of evolved features and comparison against raw RS with 1-NN

RS	Island	Rec. rate (%) VAL	Rec. rate (%) TEST	Evolved Features	Rec. rate (%) Raw RS and 1-NN TEST
RS43	1	83.11±1.32	90.47 ± 0.94	31.10±1.52	90.13
	2	81.94±4.40	89.85 ± 3.08	31.20±1.23	
	3	82.44±4.42	90.04 ± 1.36	30.00±2.26	
RS90	1	85.25±2.05	90.37 ± 1.55	31.20±1.23	91.56
	2	83.67±3.52	89.94 ± 1.25	31.10±0.99	
	3	82.21±10.99	89.19 ± 6.84	30.10±2.76	
RS91	1	79.67±5.02	89.71 ± 1.08	31.20±1.62	91.04
	2	82.60±0.94	89.89 ± 0.93	29.80±2.20	
	3	77.61±12.61	81.11 ± 5.06	31.80±0.42	
RS95	1	73.52±3.17	83.13 ± 3.29	31.80±0.42	84.92
	2	72.33±3.45	89.92 ± 2.04	30.50±2.37	
	3	72.47±3.78	83.50 ± 1.97	30.90±2.47	
RS96	1	75.75±3.48	85.14 ± 2.82	30.60±1.71	91.56
	2	76.72±1.15	89.94 ± 1.25	30.40±1.84	
	3	76.45±1.08	89.19 ± 6.84	31.40±0.96	

In addition, Table 6.6 presents the number of evolved features and the recognition rate of raw RS and 1-NN classifiers. There is a big difference between recognition rates in validation and test. Recall that during the evolution (and therefore the validation) the classifier used is an MDM and in the test phase a k-NN is used. The test recognition rates for each island are close but inferior to those of raw RS and 1-NN classifiers. The average number of evolved features is very high, reflecting the complexity of the classification problem. Each of the

evolved features is a formula of the raw features of the specific RS. In Section 6.3.5.8 we show an analysis of the utilization of raw features for each evolved classifier.

6.3.5.3 Over-fitting analysis and finding the best solution in validation

The global validation procedure is used to analyze the impact of over-fitting on the optimization of base classifiers and to find out the best solution in validation and apply it to the test data set. When global validation is applied, we mapped every individual generated during the evolution into the validation data set and find the best. Figure 6.16 shows this mapping at the end of the evolution process (11 generations for each of the 32 evolved features) for random subspace RS90. The figure displays the error rate (= 100% - recognition rate (%)) for each generation and feature. Each point in the population is represented as a red diamond. In this case there are 32 clouds of points corresponding to 32 evolved features. For each feature the cloud is more or less wide, according to the individuals' sizes (tree sizes). This allows us to show the diversity of individuals in the same generation or feature. In each cloud there is a green diamond that corresponds to the best individual in validation and a black dot that corresponds to the best individual in optimization mapped into the validation data set. Hence we can see that best individuals in optimization are not necessarily the best in validation. It means that these individuals (best in optimization) are not the best individuals to generalize. The difference in performance between these two individuals reflects the occurrence of over-fitting. The purple ellipse shows the presence of over-fitting for some evolved features.

Figure 6.17 is a zoom of Figure 6.16 in the ellipse region. We can see that evolved feature 8 has no over-fitting because the best individual in optimization mapped into validation (black spot) is located in the same place as the best individual in validation (green diamond); that is, individuals have the same performance and the same tree size. Evolved features 9, 10 and 11 have a different behaviour and the over-fitting is detected. This is made evident by the different location of the best individual in validation and the best in optimization mapped into validation, as detailed in Figure 6.17.

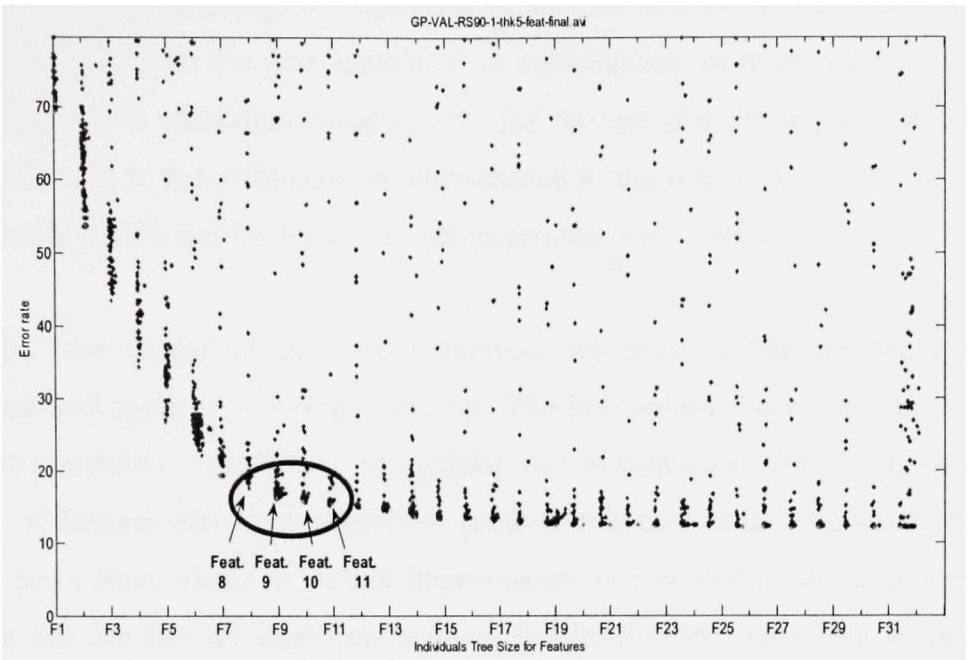


Figure 6.16 ***Over-fitting analysis for evolved RS90.***

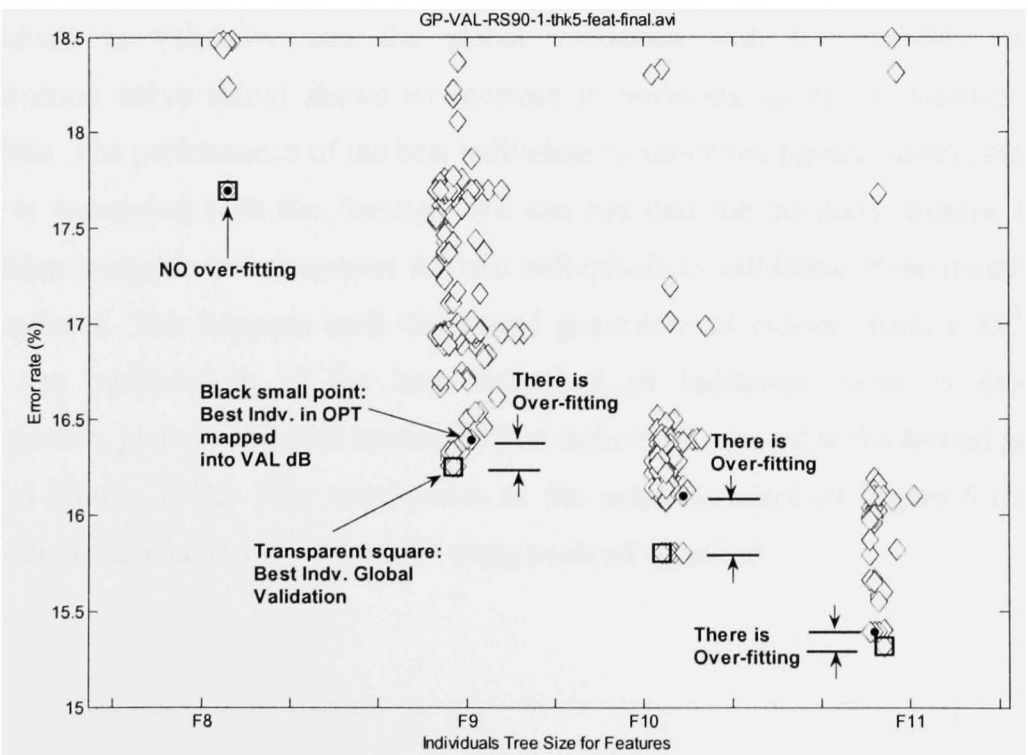


Figure 6.17 ***Over-fitting for different evolved features RS90.***

The amount of over-fitting, as explained in Section 5.3, is measured as the difference in performance between the best individual in optimization mapped onto the test set and the best individual in validation mapped onto the test set (for the same evolved feature). For evolved feature 9, the difference in performance in the test set is 0.91%; for feature 10 the difference is 0.65% and for feature 11 the difference is of 0.36%.

Once the base classifiers have been evolved, we have to find out the best solution in validation and apply it to the test data set. The best solution corresponds to the individual from the population with highest recognition rate in validation. We also have to find out the number of features that were required to produce this individual. The auxiliary file indicates the last point from where no further improvement in recognition rate is produced. In Figure 6.17 we can see that for each new evolved feature the best individual in validation (green diamond) has a lower error rate, so that the stopping point has to be located afterwards. All this information is collected by the global validation procedure. Figure 6.18 shows a detail of the recognition rate of the best individuals for each generation in optimization, the best individuals in validation and the global validation with the auxiliary archive. The optimization curve (blue) shows an increase in performance as the number of features increases. The performance of the best individual in validation (green curve) changes but the trend is increasing with the features. We can see that the auxiliary archive from global validation (magenta color) updates the best individuals in validation at the moment that they are produced. This happens until the second generation of evolved feature 25^{th}. From this point the performance of the best individual in validation starts to deteriorate, in consequence, global validation retains the best individual selected at the second generation of evolved feature EF25. This corresponds to the point indicated in Figure 6.18. The same procedure is applied to each island for every evolved classifier

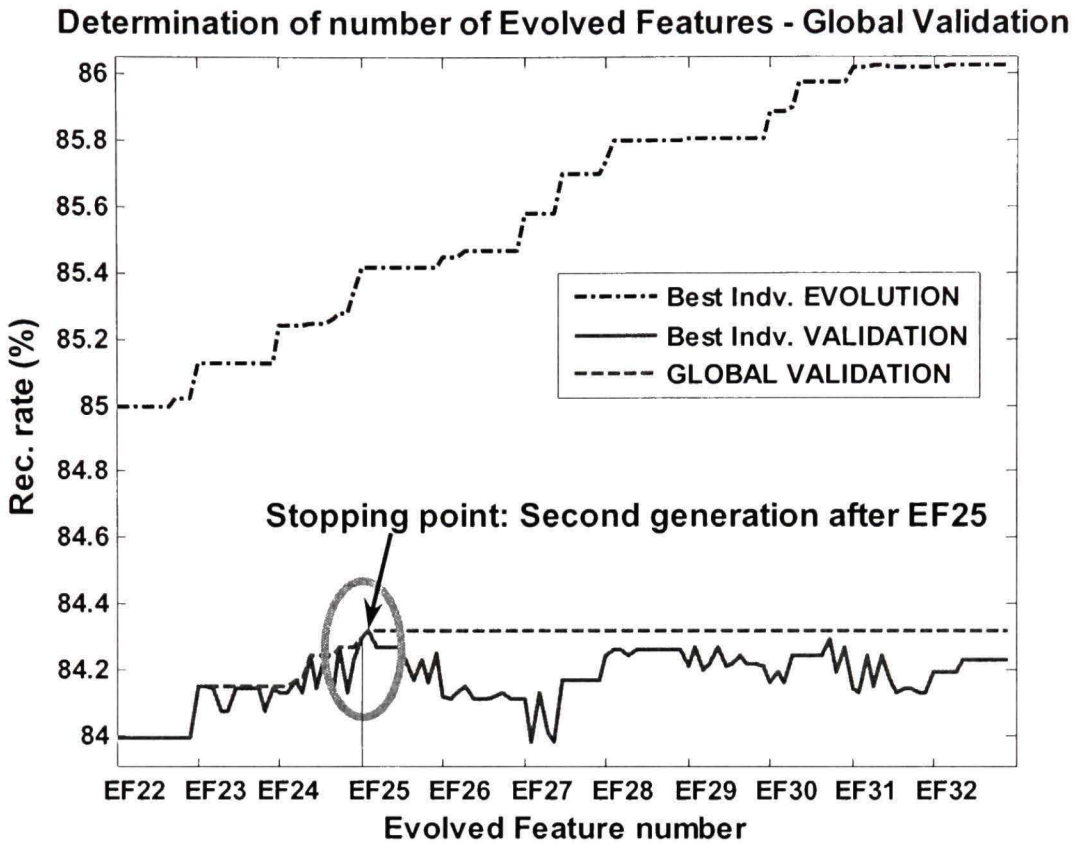


Figure 6.18 ***How to get the number of features using global validation procedure.***

Table 6.7 shows the average recognition rates of the best solution in optimization mapped into the validation data set and the best solution recorded in the auxiliary archive during the global validation procedure. Average differences reflect the existence of over-fitting. Differences in recognition rate are not big with a maximum of about 0.29%.

Solutions found are applied to the test data to obtain the recognition rates in test as well as the votes of each evolved classifier. It is worth to mention that each solution is a composition of all solutions from evolved feature one until the last feature found with the global validation method. For instance, if stopping point is at feature 25, the solution is $EF_1 \oplus EF_2 \oplus \ldots \oplus EF_{25}$. Votes for all 25 features are also recorded.

Table 6.7

Comparison of Rec. Rates in validation: mapping from OPT and VAL

RS	Island	Rec. rate (%) Best OPT in VAL	Rec. rate (%) VAL	Difference Rec. Rate
RS43	1	82.98±1.33	83.11±1.32	0.13±0.09
	2	81.78±4.43	81.94±4.40	0.16±0.08
	3	82.30±4.45	82.44±4.42	0.14±0.08
RS90	1	85.10±2.14	85.25±2.05	0.16±0.15
	2	83.42±3.46	83.67±3.52	0.25±0.19
	3	82.09±10.94	82.21±10.99	0.12±0.12
RS91	1	79.53±4.98	79.67±5.02	0.14±0.12
	2	82.30±1.04	82.60±0.94	0.29±0.24
	3	77.45±12.56	77.61±12.61	0.16±0.14
RS95	1	73.41±3.10	73.52±3.17	0.10±0.09
	2	72.16±3.43	72.33±3.45	0.18±0.12
	3	72.27±3.97	72.47±3.78	0.19±0.27
RS96	1	75.55±3.49	75.75±3.48	0.21±0.15
	2	76.53±1.18	76.72±1.15	0.19±0.13
	3	76.28±1.10	76.45±1.08	0.17±0.12

As shown in Figure 6.17, the over-fitting varies along with the evolved features, so it can decrease or increase from an evolved feature to the next. Each new evolved feature is generated with a new evolution, which can be interpreted as a new search space because the reference from which the new evolution starts considers the result to which it has arrived in previous evolved features. Since the evolution is re-started at each new evolved feature, the population that performs the search is composed as well, of new individuals. As a result, we can interpret that the over-fitting generated during the evolution of any feature is not "fully carried" to the next evolution (next feature). The degree of influence of over-fitting from previous evolved features into the new one is minimized as evolution in Bot's uses a

restrained number of generations for each evolved feature (11 generation in our case). As a result, the possibility of memorization of the data set is minimized because the optimization period is short. We can summarize saying that Bot's algorithm implements itself a way to find better solutions or solutions that generalize better for each new evolved feature. This means, the method provides a mechanism to control the over-fitting by changing the searching space and the population for each new evolved feature and using evolutions for a small number of generations.

In the following sections we present the results obtained when using various types of members of the ensemble and different combination function: ensembles of best islands, ensembles of intermediate ensembles and ensemble of all islands using simple majority vote (SMV) or weighted majority vote (WMV).

6.3.5.4 Ensembles of the best island from each evolved classifier

We take the best island for each evolved classifier and generate an ensemble by combining them with simple majority vote (SMV) as shown in Figure 6.19. This process is repeated 10 times.

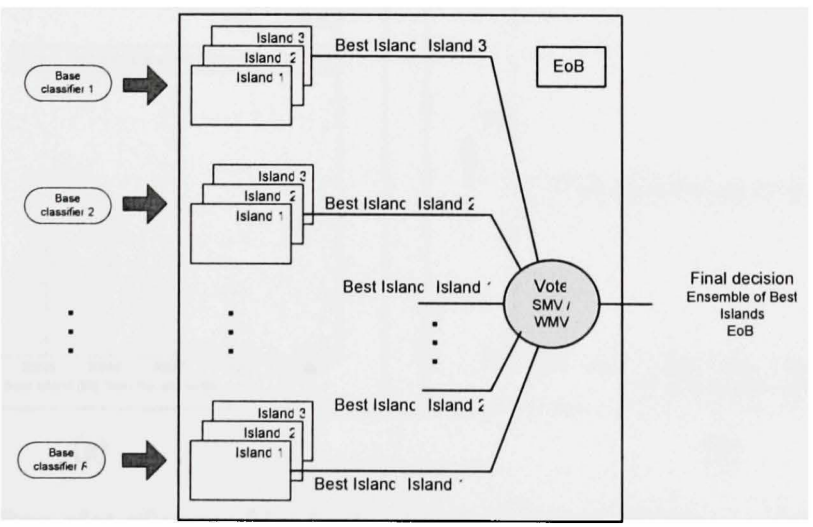
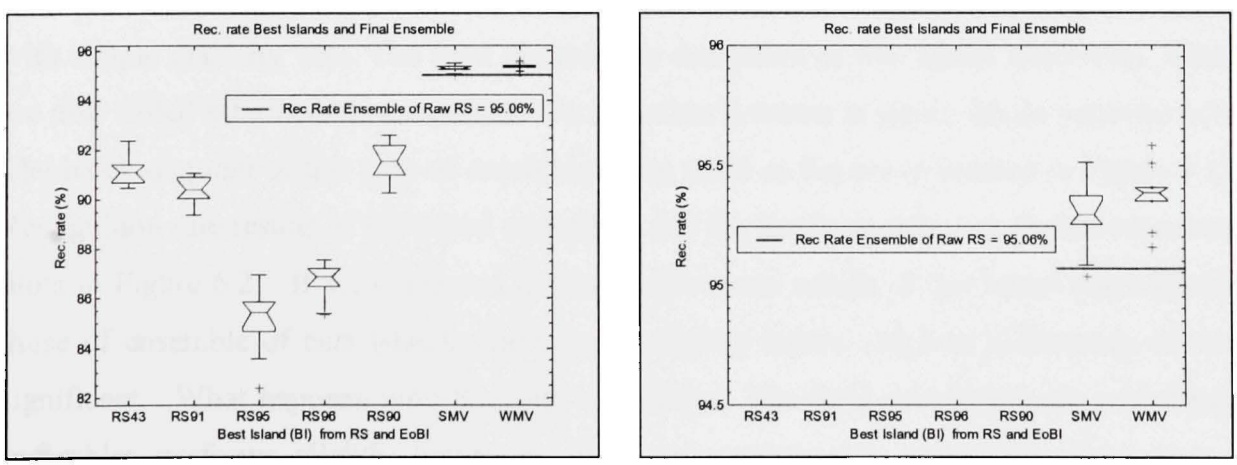


Figure 6.19 ***Diagram of formation of Ensemble of Best Islands EoBI.***

Figure 6.20(a) shows a box-plot diagram of the best islands for every evolved classifier and the resulting ensemble of Best Island (EoBI). This diagram represents the distribution of recognition rate values for each evolved classifier and the ensemble, for all repetitions. The recognition rate of the ensemble of 5 evolved classifiers is a considerable improvement if we compared it to the performance of each of its components. The mean recognition rate of the ensemble of best islands from evolved classifier (EoBI) with simple majority vote (SMV) is 95.28% ± 0.14 meanwhile the recognition rate of the ensemble of raw RS and *1*-NN classifier is 95.06%. To facilitate the comparison, this level has been indicated in Figure 6.20(a). As Figure 6.20(a) shows the range of different values for the ensemble of evolved classifiers is very small, so this ensemble consistently outperforms that of 1-NN classifiers. The right-most part of Figure 6.20 shows the ensemble EoBI with weighted majority vote (WMV). The mean recognition rate value for this case is 95.37% ± 0.12. Performance in both cases is very similar. Recognition rate values generated with weighted majority vote are less spread than with simple majority vote. Figure 6.20(b) is a zoom of part (a) that reflects the difference recognition levels between the original ensemble of 1-NN, EoBI(SMV) and EoBI(WMV). Results are reviewed in Table 6.8 above that shows the improvement obtained in recognition rate. In this case, all the ensembles have a cardinality of 5 base classifiers.



(a) (b)

Figure 6.20 ***Box-plot of ensembles built with the best island from each base classifier.***

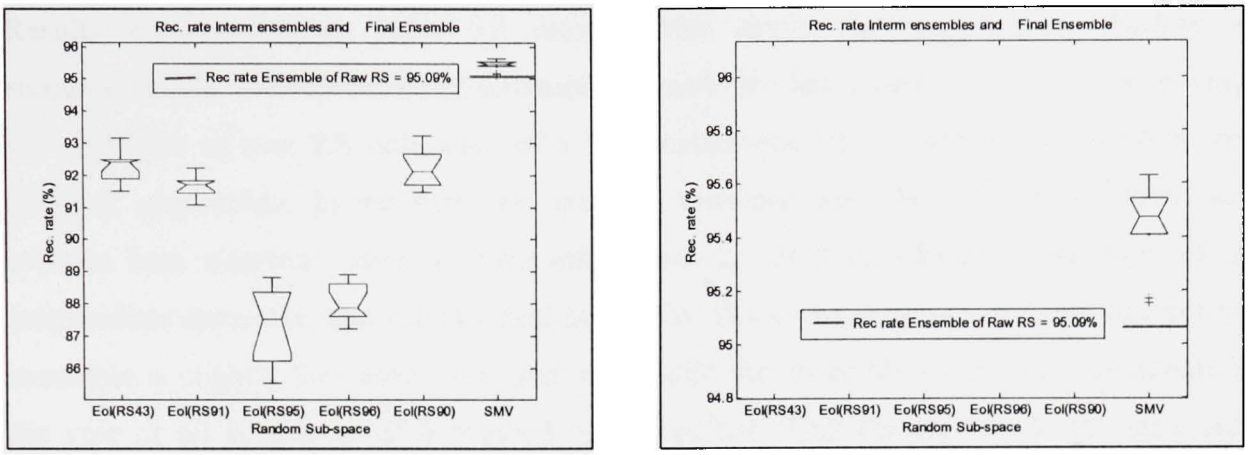
Table 6.8

Recognition rate (average and standard deviation) results of Ensemble of Best Islands fusion with simple and weighted majority vote, including a comparison to ensemble of raw RS

RS	Average Rec. rate (%) and standard deviation			Card. Ensemble	Rec. rate (%) Ensemble of Raw RS
	Single Best Island	Ensemble SMV	Ensemble WMV		
RS43	91.15±0.57	95.28 ± 0.14	95.37 ± 0.12	5	95.06
RS90	91.53±0.76				
RS91	90.41±0.55				
RS95	85.07±1.32				
RS96	86.66±0.77				

6.3.5.5 Ensembles of intermediate island ensembles

For each evolved base classifier we generate an island ensemble. This island ensemble is created with the three islands used during the optimization process and they are combined with simple majority vote. The final ensemble is composed of five island ensembles, which are also called intermediate ensembles. Combination function is again simple majority vote. The block diagram of this type of ensemble is the same as the one presented in Figure 6.12. Recognition rate results of the island ensembles and the final ensemble are presented as box-plots in Figure 6.21. If we compared the recognition rate values of the island ensembles to those of ensemble of best islands, the latter is slightly better. Anyhow differences are not significant. What happens with the final ensembles? The final ensemble built with island ensembles performs slightly better. Its mean recognition rate is 95.43%. Even though recognition rate ranges of the ensemble elements are lower, the island ensembles generate more "diverse" opinions (or votes) than those of best islands. This is clear as in the case of ensemble of the best island, the vote from the best island is the only information that comes from the evolved classifier. Since the island method is based on different searching spaces



(a) (b)

Figure 6.21 ***Box-plot of island ensembles and the final ensemble.***

covered by each island, other islands from the same RS having lower recognition rates still provide useful information for the decision of the final ensemble. Figure 6.21(b) presents a detail of the ensemble range of values in comparison to the ensemble level obtained with raw RS and 1-NN classifier. This figure shows the improvement in recognition rate obtained with the island or intermediate ensemble.

Table 6.9

Recognition rate (average and standard deviation) results of Ensemble of Intermediate ensemble, fusion with simple majority vote, including a comparison to ensemble of raw RS

RS	Rec. rate (%) and standard deviation		Card. Ensemble	Ensemble of Raw RS	
	Intermediate ensemble	Ensemble SMV		Rec. rate (%)	Card. Ensemble
RS43	92.30±0.52	95.43 ± 0.15	15	95.06	5
RS90	92.25±0.58				
RS91	91.66±0.34				
RS95	87.30±1.12				
RS96	88.00±0.59				

Results are reviewed in Table 6.9 above, which shows the improvement obtained in recognition rate. The ensemble of intermediate ensemble has a cardinality of 15 meanwhile the ensemble of raw RS only uses of 5 base classifiers. Up to here we have taken two different approaches. In the first one, we only consider one island (the best) from each evolved base classifier; later we take information of all three islands in the form of an intermediate ensemble also called island ensemble. The average recognition rate of the final ensemble is slightly increased. Next step is to build the ensemble taking into consideration the vote of all islands of each evolved base classifier. This corresponds to the third step mentioned before.

6.3.5.6 Ensembles of all islands from each evolved classifier

In this case we build directly the final ensemble considering the vote of all islands of each evolved base classifier. Then we have an ensemble of 15 evolved base classifiers as depicted in Figure 6.22. Figure 6.23 is the box-plot of the recognition rate for each island and the resulting ensembles when using simple and weighted majority vote.

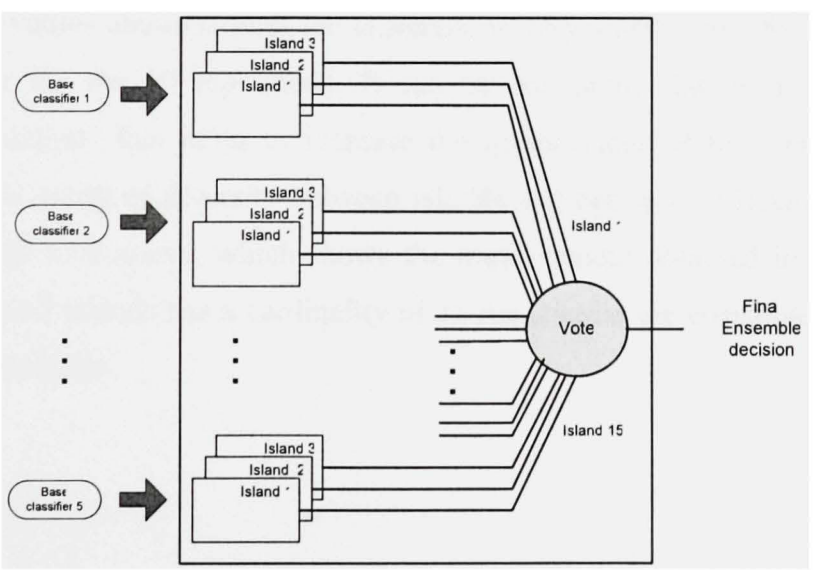


Figure 6.22 ***Block diagram of ensembles of all islands.***

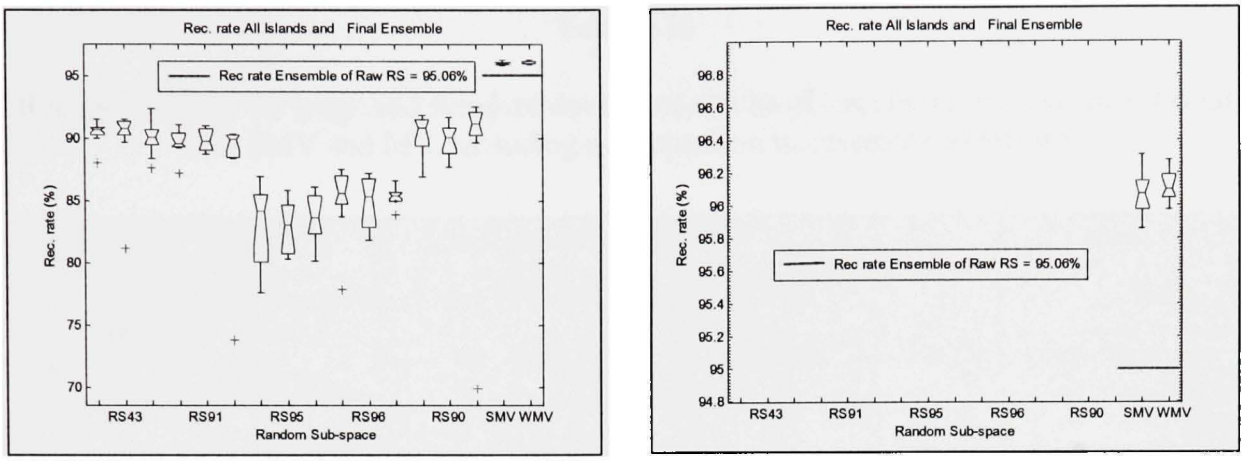


Figure 6.23 ***Box-plot of ensembles composed of 15 islands***

Each three consecutive columns correspond to the recognition rate values of a particular evolved classifier (for instance columns 4, 5 and 6 are the three island of random subspace RS91). Last two columns correspond to the recognition rate values achieved by the ensemble when using SMV and WMV respectively. In this case, the ensemble's mean performance reaches 96.06% ± 0.12 for SMV and 96.10% ± 0.10 for WMV. Again, dispersion of recognition rate values obtained with the ensemble is very small. Therefore, their responses are very similar for the 10 repetitions. It can be concluded that every island provides "different information" that helps to increase the performance of the ensemble. Different information is the result of diversity between islands and between different RS. Results are reviewed in Table 6.10 above, which shows the improvement obtained in recognition rate. The ensemble of all islands has a cardinality of 15 meanwhile the ensemble of raw RS only uses of 5 base classifiers.

Table 6.10

Recognition rate (average and standard deviation) results of Ensemble of all islands, fusion with SMV and MV, including a comparison to ensemble of raw RS

RS	Island	Ensemble of all islands				Ensemble of raw RS	
		Average Rec. rate (%) and std. deviation			Card.		
		Islands	Ensemble SMV	Ensemble WMV		Rec. rate (%)	Card.
RS43	1	90.47 ± 0.94	96.06±0.12	96.10±0.10	15	95.06	5
	2	89.85 ± 3.08					
	3	90.04 ± 1.36					
RS90	1	90.37 ± 1.55					
	2	89.94 ± 1.25					
	3	89.19 ± 6.84					
RS91	1	89.71 ± 1.08					
	2	89.89 ± 0.93					
	3	81.11 ± 5.06					
RS95	1	83.13 ± 3.29					
	2	89.92 ± 2.04					
	3	83.50 ± 1.97					
RS96	1	85.14 ± 2.82					
	2	89.94 ± 1.25					
	3	89.19 ± 6.84					

6.3.5.7 Review of results and analysis of smaller ensembles

Table 6.11 reviews the results of this section and compares them to ensembles built with 1-NN classifiers only based on RS. Best values are shown in bold. The ensembles built with evolved classifiers outperform ensembles based on 1-NN classifiers, in more than 1%, for the

two fusion methods tested: simple and weighted majority vote. The average number of evolved features required to achieve these recognition rate levels is very close to the original number (32).

Table 6.11

Review of results selected ensemble built with evolved classifiers

Fusion method	**Best Island**	**Island Ensemble**	**All islands.**	**1-NN**
	Recognition rate (%) and standard deviation			**Rec. rate (%)**
SMV	95.28 ± 0.14	95.43 ± 0.15	**96.06** ± 0.12	95.06
WMV	95.37 ± 0.12		**96.10** ± 0.10	

Table 6.12

Recognition rates for ensembles of 4 evolved classifiers (12 in total)

Combinations	RS43 (%)	RS91 (%)	RS95 (%)	RS96 (%)	RS90 (%)	Rec. rate (%)
1	X	X	X	X		95.58 ± 0.12
2	X	X	X		X	**96.09**± 0.13
3		X	X	X	X	95.60± 0.12
4	X	X		X	X	95.93± 0.11
5	X		X	X	X	95.66±0.11

From Figure 6.20 and Figure 6.23, we see that some classifiers of the ensemble have very low performance in comparison to the others. In this subsection we investigate the impact of removing one evolved classifier from the ensemble. We verify all possible combinations of 4 evolved classifiers out of 5 available. Decision of the ensemble is weighted majority vote. Figure 6.23 shows that RS95 has the lowest performance of all base classifiers considered, followed by RS96. Table 6.12 shows the five different combinations of ensembles of 4

evolved classifiers out of 5 (12 out of 15 evolved classifiers considering the islands). The evolved classifiers (with their 3 islands) that are included in the ensemble are indicated with an X in the corresponding row. For instance, the ensemble with evolved classifiers {RS43, RS91, RS95, RS90} generates a mean recognition rate of 96.09%. We conclude that RS95 is very useful in the ensemble, that is, it provides diversity to the ensemble so the recognition rate is improved. The recognition rate obtained with the ensemble of {RS43, RS91, RS95, RS90} is better than the ensemble of five base classifiers considering all islands in each one and with a simple majority vote as combination function. In turn, its performance is inferior to the weighted majority vote of the same ensemble {RS43, RS91, RS95, RS90}. But definitely, results from Table 6.12 show that ensembles built are very performing and with a reduced number of base classifiers.

We reproduce in Table 6.13 the recognition rates of the best selection methods on the test set used in (Tremblay, 2004). We show in bold, the optimized ensembles with better performance than our best ensembles. The ensembles obtained with the method of feature creation for ensembles have recognition rates very close to the results in Table 6.13. The main difference is the number of base classifiers. The smallest comparable ensemble from Table 6.13 has 31 base classifiers and our ensembles only have 9 or 12 evolved classifiers (when considering all the islands). Therefore, it is very promising that adding new base classifiers could increase even more the performance of the ensemble. Furthermore, the resulting ensembles have controlled at maximum the over-fitting by means of the global validation mechanism. The following section provides an analysis about the raw feature utilization. From Table 6.14, we can calculate that each evolved classifier, in average, utilizes 27.24 raw features. As a result, the whole ensemble would be composed of an average of 27.24x5=137 raw features. The ensemble of raw RS and 1-NN classifiers uses 32x5=160 raw features. Therefore, proposed ensemble of evolved classifiers uses, in average, only the 85.62% of raw features used by raw RS.

Table 6.13

Results from optimization of ensembles with GA
Extracted from (Tremblay, 2004)

Method	# Classifiers	Rec. rate on TEST
k-NN (132 feat.)	1	93.34%
Ens. 100 k-NN	100	**96.28%**
MLP (132 feat.)	1	95.27%
Ranking	76	**96.26%**
Simple GA	31	**96.41%**
NSGA-e	24	**96.40%**

6.3.5.8 Feature selection and feature creation

We have mentioned before (Section 5.4) and revealed with the results shown (Chapters 4, 5 and 6) that Bot's method creates a compact representation of a problem that yields similar or better results in classification than the initial representation. This compact representation is a set of evolved features, each one being a formula of the input raw features and a functions set (+, -, *, /, sin, cos, log and exp). The compact representation is one of the advantages of Bot's method. The subset of raw features to be used depends in some degree of the problem itself and the ability of the method presented is to discover which is the best combination between the formulas generated and the most discriminating raw features to use instead of handling all of them. As shown in Chapter 5, the proposed method is able to select the raw features to use in conjunction with the generated formulas. That is, the algorithm performs at the same time feature creation and feature selection. Effectiveness of any raw feature can be analyzed from two perspectives that are complementary. In the first one, we ask if the raw feature used or not in the evolved features and, in the second perspective, if the raw feature is use, how important is it to from the performance point of view. In the former we are interested to know if the raw feature is used independently of the number of times it appears in the solutions. In

this case we pay attention to the feature selection side. In the latter perspective, we are interested in recording how many times the raw feature is used, which may be related to the relative importance of the selected raw feature.

Table 6.14 presents the average and standard deviation results of the raw feature utilization for five evolved RS, considering in each case the 3 islands. Evolved classifiers considered in this case are those used in the ensemble of Section 6.3.5.4. The input for the evolution of each RS was a set of 32 raw features. The dispersion of number of evolved features for each RS was presented in Figure 6.15. In average, 30 evolved features (90 in total considering all three islands) are required to represent an evolved RS. But how many raw features are needed? From Table 6.14 we calculate that in average 27.24 ± 0.87 different raw features out of 32 were used in the final solutions. It means that the cardinality of the RS (32) is appropriate and going down to RS of 16 raw features could not be very representative of the original data set. We now analyze which are the most important raw features by looking their presence through 10 replications of the evolution of RS classifiers.

From the selection perspective we analyze in how many of the 10 replications, a raw feature is used at least one time in the evolved features. In fact, it means that this particular raw feature has to be "selected" to build the evolved classifier. If a raw is consistently used through most of the repetitions (or all of them), we know that this raw feature is an important component of the evolved features (solutions). In this way, we establish in average how many different raw features are used to generate the evolved features. This information is presented in Table 6.14 as raw feature utilization.

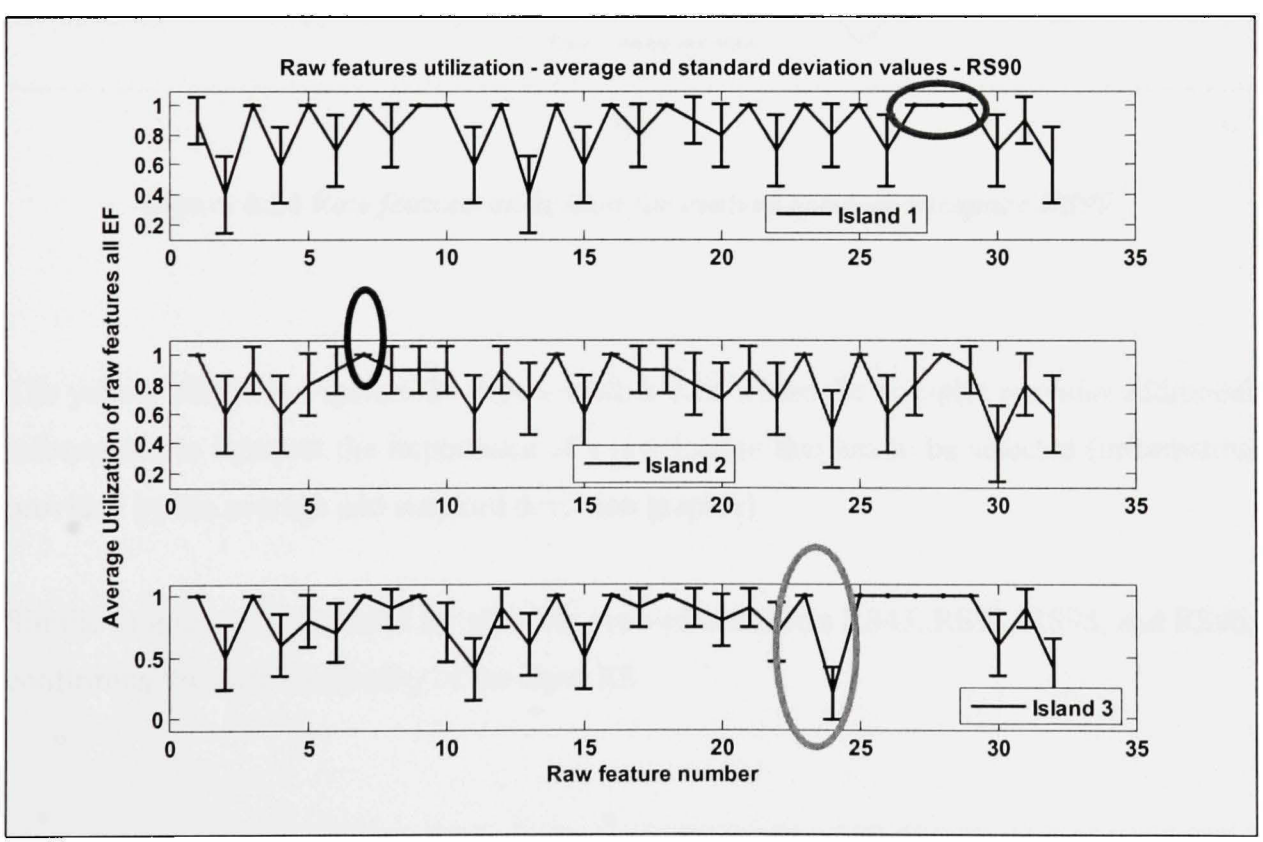
Table 6.14

Raw features utilization for evolved RS

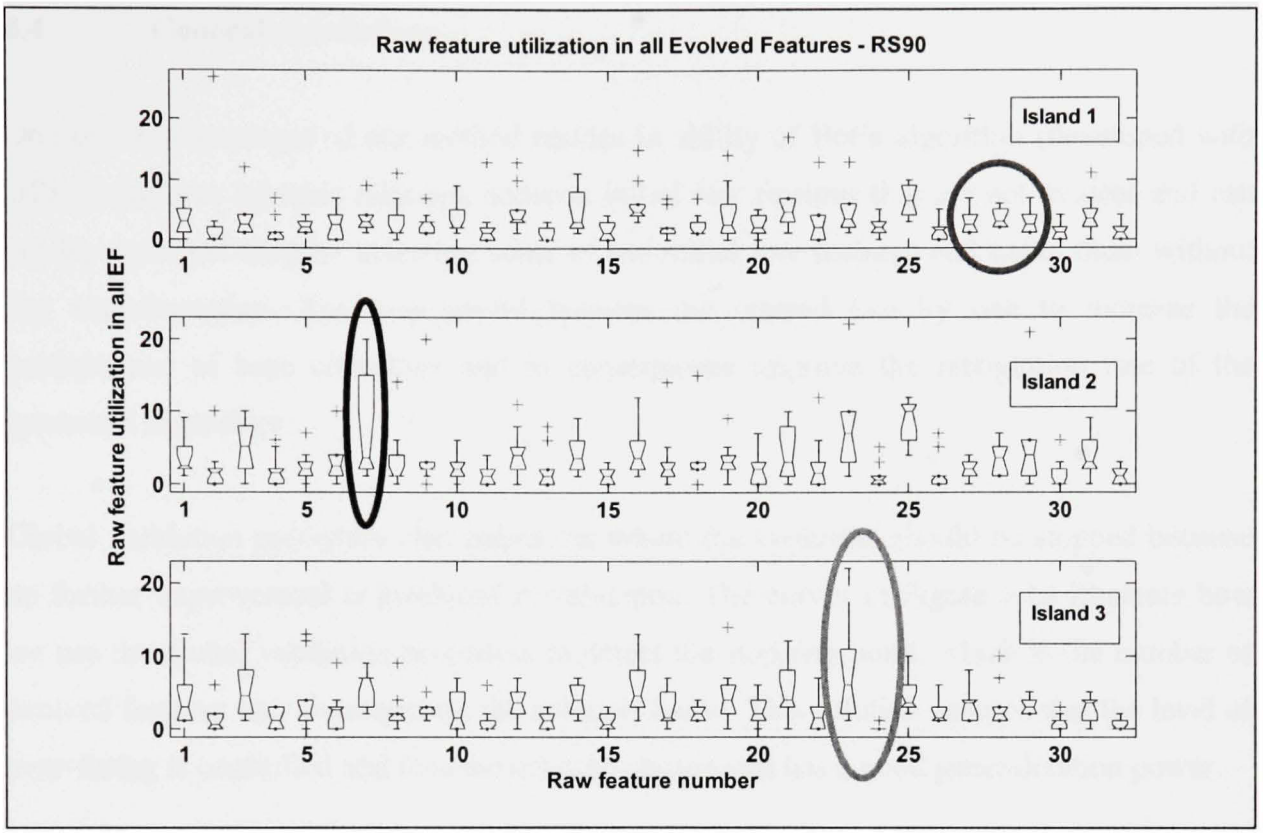
Random Subspace	Island	Features	Raw Features Utilization
		#F*ave* ± Std(#F)	RawF*ave* ± Std(RawF)
RS43	1	31.10±1.52	27.70±2.49
	2	31.20±1.23	28.00±1.24
	3	30.00±2.26	27.50±2.01
RS90	1	31.20±1.23	26.90±1.66
	2	31.10±0.99	25.80±2.25
	3	30.10±2.76	25.90±1.79
RS91	1	31.20±1.62	27.70±0.82
	2	29.80±2.20	26.60±1.95
	3	31.80±0.42	28.70±1.25
RS95	1	31.80±0.42	27.40±2.99
	2	30.50±2.37	26.90±2.23
	3	30.90±2.47	28.40±2.60
RS96	1	30.60±1.71	27.00±2.35
	2	30.40±1.84	26.20±1.47
	3	31.40±0.96	27.90±1.66

Figure 6.24(a) shows the average and standard deviation of raw feature utilization for each raw feature from the "selection" perspective (it shows if raw feature was selected at least one time in each repetition). Figure 6.24(b) shows a box-plot of the total number of times each raw feature is used in the evolved features. This figure makes the difference between "selected" features, to emphasise the most important raw features from classification perspective. It shows how many times a raw feature was part of the evolved features.

Figure 6.24 exemplifies three different cases (ellipses to point out them). For instance raw features 27, 28 and 29 for island 1 as indicated in the red ellipse have an average raw utilization of 1 and a zero standard deviation because they are selected in all 10 repetitions as part of the evolved features formulas. Figure 6.24(b) reveals that these raw features are used just a few times in each repetition, because the median is very low and the inter-quartile distance is small. The blue ellipse in Island-2 shows a different situation for raw feature 8. Even though the average raw utilization is also 1 with zero standard deviation (Figure 6.24 (a)), the box-plot reveals that this raw feature is highly used in some of the repetitions and not just a few times. We conclude that raw features 27, 28 and 29 for Island-1 have to be selected but raw feature 8 must be selected and it is an important component of the solution. We can go further saying that in the individuals (trees) of the solution, the raw feature 8 is part of a building block.



(a)



(b)

Figure 6.24 ***Raw feature utilization for evolved random subspace RS90.***

The yellow ellipse in Figure 6.24 shows another case where the box-plot provides additional information to interpret the importance of a raw feature that has to be selected (information provided by the average and standard deviation graphic).

Similar graphics are produced for all other evolved classifiers RS43, RS91, RS95, and RS96, confirming the right cardinality of the input RS.

6.4 General conclusions

One of the advantages of our method resides in ability of Bot's algorithm (developed with GP) to discover intrinsic relations between initial raw features that are not evident and can not be exploited only by selecting some of the initial raw features and using them without any transformation. The new useful features are created one by one to increase the performance of base classifiers and in consequence improve the recognition rate of the generated ensembles.

Global validation procedure also points out where the evolution should be stopped because no further improvement is produced in validation. The curves in Figure 6.18 illustrate how we use the global validation procedure to detect the stopping point, which is the number of evolved features and consequently the solution found. This solution ensures that the level of over-fitting is controlled and thus the solution chosen still has a good generalization power.

Since the method only used a few generations for each feature, the total computational time is lower than conventional applications of GP. In addition, the reinsertion of an initial population randomly created at the beginning of the evolution of a new feature improves the diversity of the population. Moreover, the application of island method during the evolution emphasizes the search over different spaces generating more performing solutions and at the same time increasing the diversity and avoiding the stagnation of the evolution.

We proposed two different methods to create EoC. Each base classifier corresponds to an evolved classifier built with the improved Bot's algorithm. In the first one, we combined the votes from each base feature by feature. The results show that performance is improved along with the addition of new features. For ensembles of five or more classifiers, our approach based on evolved classifiers, outperforms the corresponding ensembles of 1-NN classifiers. Hence, the optimization of the classifiers adds useful information to build more performing ensembles. What is more, a certain level of performance is attained after some features (this varies from 12 to 15 features). As a result, we can build EoC to assure certain

performance with the minimum number of evolved features. Thus reduces the complexity of the ensemble without reducing the performance. We obtain an average performance of 96.14% for an ensemble of 9 base classifiers evolved for 21 features meanwhile the corresponding ensemble of 1-NN classifiers attains 95.57% in the test set, in short we outperforms in 0.57%. When we compare our results with the overall best results reported in (Tremblay, 2004) and reproduced in Table 6.13, the performance of our approach is only 0.27% less than the best ensemble of 1-NN classifiers. Solutions reported in (Tremblay, 2004) were searched with simple GA and with NSGA in a pool of 100 1-NN classifiers. In the first case, Tremblay's solution had 31 classifiers and in the second case, 24 classifiers. In our approach 27 classifiers were used to build the ensemble. In addition, the input RS used have different levels of performance: very good, medium and very poor performances. Finally, we only use 21 evolved features to attain this level of performance.

The second method proposed to create the ensembles was based on finding for each base classifier the maximum number of evolved features before over-fitting the optimization data set. The base classifiers have then different number of evolved features but each one provides the best recognition rate controlling at maximum the over-fitting. We take a specific ensemble selected in (Tremblay, 2004). The ensembles built used simple majority vote and weighted majority vote and we obtain average recognition rate of 96.06% and 96.10% respectively. The corresponding ensemble of 1-NN classifiers obtained 95.067% in the test set. We outperforms for more than 1%. Once more, our approach makes evident that the features built for ensembles add useful information that makes them very promising. When comparing our results to the best ensembles reported in (Tremblay, 2004), we only are 0.30% down but instead of using 24 or 31 classifiers, our ensemble is composed of only 15 classifiers (3 from each evolved classifier).

The proposed method to evolve classifiers for ensembles creates a compact representation of a problem in such a way that his representation yields similar or better results in classification than the initial representation. In addition, we have shown that the evolved

features used some of the raw features. The subset of raw features to be used depends to some degree of the problem itself and the ability of the method presented is to discover which is the best combination between the formulas generated and the most discriminating raw features to use instead of handling all of them. That is, the algorithm performs at the same time, feature creation and feature selection.

CONCLUSIONS AND RECOMMENDATIONS

This thesis proposed a re-engineering method of base classifiers for ensembles. The original base classifiers, used as input to our system, were built by the RS method, so each one has a partial representation of the recognition problem. The re-engineering process applied to each base classifier consisted in creating "evolved features" from its partial representation. The creation of evolved features consisted in discovering intrinsic relations between the original features, called raw features, in such a way that the resulting evolved classifiers have a better performance in classification. The combination of evolved base classifiers created ensembles with higher recognition rates. This approach was tested in a complex pattern recognition problem: recognition of isolated handwritten digits from NIST-SD19 data set.

The generation of intrinsic relations between raw features was made using GP, which is capable of finding the appropriate structure and values of the solution. The creation of solutions in such a way is not possible with other optimization techniques.

A series of different experiments were run to test each stage of our system. Initially we tested the ability of our method to represent some UCI data set in a condensed set of evolved features with good recognition rate levels. Bot's method of creating evolved features one at a time was successfully tested over 16 data sets from UCI repository by means of a wrapper approach based in GP and a classifier. Two classifiers were tested with positive results in UCI data sets: nearest neighbour (k-NN), minimum distance to means (MDM). Recognition rate calculated by classifiers was the fitness measure used to guide the search of the GP algorithm. In average, the UCI data sets analyzed were well represented by 3 to 4 evolved features with recognition rates comparables to those obtained when the whole set of raw features is used.

In a second set of experiments, global validation procedure was incorporated and it provided representations with higher generalization power as shown with some of the UCI data sets tested. This improvement was possible because of the control of over-fitting during the

optimization process. Furthermore, the auxiliary archive used in global validation allowed the correct identification of the stopping point during the evolution before going into over-fitting. We found that the evolutionary process generates over-fitting towards the optimization data set used, but Bot's algorithm implements itself a way to find better solutions or solutions that generalize better for each new evolved feature. This means, the algorithm provides a mechanism to control the over-fitting by changing the searching space and the population for each new evolved feature and using evolutions for a small number of generations. The two first factors reduce the chances of getting trapped in a local maximum during the evolution. And the small number of generation during each evolution decreases the possibility of over-fitting. On top of that, Bot's algorithm combined with global validation procedure permit to precisely establish the moment to stop the evolution and the addition of new evolved features. The analysis of results of the global validation procedure points out that the influence of over-fitting due to the evolutionary process is high in some of the UCI data sets analyzed and it does not influenced that much the evolved RS analyzed from the NIST SD19 data set.

Since Bot's method only uses a few generations for each feature, the total computational time is lower than conventional applications of GP. In addition, the reinsertion of an initial population randomly created at the beginning of the evolution of a new evolved feature improves the diversity of the population. Bot's method was further improved by using a parallelization strategy based on island method. The island method provided an improvement not related to the computational time. The computation time is equivalent to a run using a panmictic population. The gain in performance resides in the parallel exploration of different zones of the searching space. What is more, the island method proved to be robust as the evolution can continue even when one of the island stops evolving (for instance, due to a problem in the processor managing running the evolution). In addition to the robust quality of island model of parallelization, the use of a combination of the outcomes from every island (called V-best solutions) showed an improvement over single best solutions. The ensemble created in such a way has a better performance than its components because of the diversity of each solution, due to the exploration of different zones in the searching space.

We proposed two different methods to create EoC and tested them in complex problem of isolated handwritten digit recognition. Each base classifier corresponds to an evolved classifier built with the improved Bot's algorithm. In the first one, we combine the votes from each evolved classifier feature by feature. The results show that performance is improved along with the addition of new features. What is more, a certain level of performance is attained after some features (from 15 to 20 features out of 32 initial features). As a result, we can build EoC to assure certain performance with the minimum number of evolved features. Thus reduces the complexity of the ensemble without reducing the performance. We obtained a performance of 96.10% for an ensemble of 9 evolved classifiers along 21 features.

The second method proposed to create the ensembles was based on finding for each base classifier, the maximum number of evolved features before over-fitting the optimization data set. The evolved classifiers have then different number of evolved features but each one provides the best recognition rate controlling at maximum the over-fitting. In this case we were able to built ensembles with performances comparable to those reported in (Tremblay, 2004), but with as few as 9 or 12 base classifiers.

Following are the **recommendations** to further improve the system:
Improve the control of over-fitting by splitting the validation data set in different disjoint partitions. The number of partitions depends on the number of evolved feature to use. Validation for the first evolved feature is made by applying the global validation procedure with the first validation partition. If over-fitting occurs during the evolution of the first feature, the best individual in optimization is replaced with the best individual found with the first validation partition in order to compute subsequent evolving features. The same procedure is repeated until the evolution of the last feature. With this mechanism the over-fitting can be reduced. The disadvantage is the stratification of the validation set into smaller partitions that may be are not as representative of the whole data set as the validation set complete. In addition, the generalization power claimed by each evolved feature based on a

validation partition depends on how representative of the whole data set is the particular partition.

Include a variant in Bot's algorithm that re-starts a new feature search when the last one was discarded because its low recognition rate improvements. During the creation of evolved features, the incremental contribution is measured and compared to a threshold to decide if it is worth to have an additional evolved feature because of its contribution. If the incremental contribution (in optimization) is not higher than the threshold, this evolved feature EF(*f*) is discarded and the evolution process is stopped. The variant is applied in case of a first discard of the evolved feature EF(*f*). Once tentative feature is discarded, a new evolved feature EF(*f*) is created in a second phase. After comparison against the threshold, two alternatives can be taken. If the gain of new EF(*f*) is greater than the threshold, this second evolved feature EF(*f*) is accepted as usual and evolution continues. If again, the contribution of the second time evolved feature EF(*f*) is smaller than threshold, then the new contribution is compared to the contribution of previous evolved feature EF(*f*). In case of being greater a last chance is given to create a definitive evolved feature EF(*f*). If the contribution of the second time evolved feature EF(*f*) is inferior to the discarded first evolved feature EF(*f*), then this second time evolved feature EF(*f*) is also discarded and the evolution is stopped. In the last case, it is clear that new attempts to create a new evolved feature will not produce a considerable gain.

Include the automatic control of genetic operator probabilities as presented in (Michalewicz and Schmidt, 2007). The mechanism counts the number of times that an operator produces an off-spring that is better than all its parents and the success ratio for each operator can be calculated as the number of improved off-springs divided over all improved off-springs. The probabilities are set as the addition of the previous probability plus the success ratio multiplied by a constant in the range of zero to one. Initial probabilities are set to small values and they will be adjusted as evolution goes. If no improved off-springs are produced, the success ratio is not defined and probabilities are not adjusted (Michalewicz and Schmidt, 2007).

Another possible improvement is to construct the ensembles of classifiers based on the W-best evolved formulas, instead of only taking the single best formula from each island and evolved classifier. During the evolution of the classifiers, the optimization phase is similar to the one presented and the global validation procedure retains in the auxiliary archive S, the W-best individuals from each sub-population evolved. At the end of evolution, the W-best evolved individuals (on the validation data set) from each sub-population are taken to be part of the ensemble of evolved classifiers. Therefore, ensembles of R input classifiers with V sub-populations will be composed by $R*V*W$ evolved classifiers. This option could improve the diversity of opinions of the ensemble when taking the decision of the samples in the test set.

We have evolved each base classifier independently. In order to consider the different information from each RS, they can be evolved sequentially. In this variant, the fitness function considers the decision of all base classifiers evolved or not. This intends to guide the searching as function of the global response of all the elements of the ensemble. A natural extension to this approach would be the evolution in parallel of all the base classifiers. These two options are just mentioned from a theoretical perspective because the computational demands would be massive and the computational time extremely long.

ANNEX I

STEADY-SATE REPLACEMENT

The concept of generation gap is related to the notions of non-overlapping and overlapping populations. In a non-overlapping population, all parents are replaced by offspring and so there is non competence between them (Sarma and De Jong, 1997). This is the conventional replacement scheme in GA and is called *generational*. In the overlapping version, parents and offspring co-exist and compete between them. This replacement scheme has been named *Steady-State* GA. Generation gap is referred as the proportion of individuals in a population that are replaced in each generation (Beasley *et al.*, 1993).

To work with the Steady-State replacement GA, selections of individuals to be parents and the individuals to be replaced (to make room for the offspring) have to be addressed. Some of the possible schemes are (Beasley *et al.*, 1993):

- Selection of parents according to fitness, and selection of replacement at random
- Selection of parents at random, and selection of replacement by inverse fitness
- Selection of parents and replacements according to fitness/inverse fitness

Depending on the GA-system and applications, a scheme or other is applied. A difference between a conventional, generational GA and a steady state GA is that population statistics are recomputed after each mating in steady state GA and new offspring are immediately available for reproduction (Beasley *et al.*, 1993). A comparison between the two replacement strategies was carried on (Sarma and De Jong, 1997). Sarma and De Jong found that that the main difference is a higher genetic drift (allele loss[1]) in steady state, especially when small population sizes are used with low generation gap values (Sarma and De Jong, 1997) (a small number of individuals replaced in each generation). Previous conclusions are based on the assumption that the steady state uses a uniform deletion. If this is not the case, the balance between exploration and exploitation is altered (Sarma and De Jong, 1997).

Analysis about the replacement mechanism has been made for GA, but it is also applicable in GP.

---

[1] In biology the concrete realization of a gene is called an allele and represents the logical entity on top of the molecular level Wagner and Affenzeller, 2005). In the context of GAs describes the basic entity that represents genetic information and forms chromosomes. For instance in binary encoding an allele can e.g. be a bit at a specific position of the individual's bit string (Wagner and Affenzeller, 2005).

ANNEX II

BOT'S ALGORITHM

Input:
Nbr_Gen : Number of generations to evolve solutions for one feature
Max_Nb_features : Maximum number of evolved features
Output:
Def_Sol : Definitive selected individual (formula) for each feature

start - Bot's algorithm
1 f=1; EF(f=1)=Φ; g =0;
2 Initialize population P(f,g)
3 Evaluate fitness(P(f,g))
4 **for** g=1 to *Nbr_Gen*
P'(f,g) = select_parents(P(f,g))
Perform genetic operations(P'(f,g)) : reprod., x-over, mut.
Evaluate fitness(P'(f,g))
P(f,g+1) = New_population(P'(f,g), P(f,g))
end for
} Run_Evol(.) (f=1, g>0)
5 **F**(f=1) = Best_Indv(P(f)) // Selected_Indv(f=0)
6 **repeat**
7 **EFprior**(f-1) = **EF**(f-1) ⊕ **EF**(f-2) ⊕ … ⊕ **EF**(1)
8 g =0
9 Initialize population P(f,g)
10 Evaluate fitness(P(f,g) , **EFprior**(f-1))
11 **for** g =1 to *Nbr_Gen*
P'(f,g) = select_parents(P(f,g))
Perform genetic operations(P'(f,g)) : reprod., x-over, mut.
Evaluate fitness(P'(f,g), **EFprior(f-1)**)
P(f,g+1) = New_population(P'(f,g), P(f,g))
end for
} Run_Evol(.) (f>1, g>0)
12 Selected_Indv(f) = Best_Indv(P(f)) // Selected_Indv(f)
13 stop = Bot_stopping(Selected_Indv(f), **EFprior**(f-1), f)
14 **if** ~*stop*
15 **EF**(f) = Selected_Indv(f)
16 **EFprior**(f) = **EF**(f) ⊕ **EFprior**(f-1) // new features to de added
17 f=f+1
18 **else**
19 **Def_Sol** = **EFprior**(f-1)= **EF**(f-1) ⊕ **EF**(f-2) ⊕ … ⊕ **EF**(1) // no more features
20 **end-if**
21 **until** *stop criterion met*
end – Bot procedure

ANNEX III

BOT'S ALGORITHM WITH VALIDATION AFTER OPTIMIZATION

Input:
Nbr_Gen : Number of generations to evolve solutions for one feature
Max_Nb_features : Maximum number of evolved features
Output:
Def_Sol : Definitive selected individual (formula) for each feature

start - Bot's algorithm
1 f=1; EF(f=1)=Φ; g =0;
2 Initialize population P(f,g)
3 Evaluate fitness(P(f,g))
4 Run_Evol (f=1, g>0)
5 **F**(f=1) = Best_Indv(P(f)) // Selected_Indv(f=0)

6 **repeat**
7 **EFprior**(f-1) = **EF**(f-1) ⊕ **EF**(f-2) ⊕ … ⊕ **EF**(1)
8 g =0
9 Initialize population P(f,g)
10 Evaluate fitness(P(f,g) , **EFprior**(f-1))
11 Run_Evol (f>1, g>0)
12 Selected_Indv(f) = Best_Indv(P(f)) // Selected_Indv(f)

13 stop = Bot_stopping(Selected_Indv(f), **EFprior**(f-1), f)
14 **if** ~*stop*
15 **EF**(f) = Selected_Indv(f)
16 **EFprior**(f) = **EF**(f) ⊕ **EFprior**(f-1) // new features to de added
17 f = f+1
18 **else**
19 **Def_Sol** = **EFprior**(f-1)= **EF**(f-1) ⊕ **EF**(f-2) ⊕ … ⊕ **EF**(1) // no more features
20 **end-if**
21 **until** *stop criterion met*
22 Change OPT dB with VAL dB
23 Evaluate fitness(P(f= *Max_Nb_features*, g= *Nbr_Gen*) , **EFprior**(f-1))
24 Best_Indv_VAL(P(f= *Max_Nb_features*, g= *Nbr_Gen*)) =
Best_Indv(P(f= *Max_Nb_features*, g= *Nbr_Gen*))
25 Change VAL dB with TEST dB
26 Calculate Rec. rate { Best_Indv_VAL(P(f= *Max_Nb_features*, g= *Nbr_Gen*)) }
end – Bot procedure with Validation after Optimization

ANNEX IV

GLOBAL VALIDATION STRATEGY (ADAPTED FROM (Radtke *et al.*, 2006))

Global Validation strategy - adapted from (Radtke *et al.*, 2006)

Input: Empty auxiliary archive S

Output: Auxiliary archive S with the best evolved individual in the Validation set for each generation

```
1   Initialize Auxiliary archive S=Ø;
2   Generate Initial population
3   Fitness measure:
4       Fitness evaluation (VAL dataset);
5       Rank Individuals
6   Update Auxiliary archive S
7   while ~non_stop
8       Generate evolved individuals : evolved(gen)
9       Fitness evaluation;
10      Rank Individuals;
11      Update Auxiliary archive S:
12          Fitness evaluation (VAL dataset);
13          Rank Individuals;
14          Update auxiliary archive S with best individual between individual from
15              generation g and generation (g-1);
16      end-Update
17      g= g + 1;
18      Verify stop_criterion;
19  end –while
```

ANNEX V

GLOBAL VALIDATION PROCEDURE APPLIED TO BOT'S METHOD

Input:
Nbr_Gen : Number of generations to evolve solutions for one feature
Max_Nb_features : Maximum number of evolved features
Output:
Def_Sol_OPT : Definitive selected individual in OPT for each feature
S : { **def. sel. individuals** $_f$ ***(gen)(VAL),*** *Feat_Val, Gen_Val* }
Rr-$_{OPT}$: Recog. Rate of Best Indv. in OPT applied to TEST dataset
Rr-$_{VAL}$: Recog. Rate of Best Indv. in VAL applied to TEST dataset

start - Bot's algorithm
1 f=1; EF(f=1)= Ø; g =0;
2 Initialize Auxiliary archive S= Ø
2 Initialize population P(f,g)
3 Evaluate fitness(P(f,g))
4 $P_{VAL}(f,g)$ = Apply P(f,g) interchanging OPT dB with VAL dB
5 Evaluate fitness($P_{VAL}(f,g)$)
6 Update S with Best_Indv_VAL($P_{VAL}(f,g)$)
7 Interchange back VAL dB with OPT dB
8 **for** g=1 to *Nbr_Gen*
9 Run_Evol (f, g>0)
10 **$P'_{VAL}(f,g)$ = Apply P'(f,g) interchanging OPT dB with VAL dB**
11 **Evaluate fitness($P'_{VAL}(f,g)$)**
12 **Update Auxiliary archive S:**
13 **if fit(Best_VAL($P_{VAL}(f,g)$) > fit(Best_VAL($P_{VAL}(f,g-1)$))**
14 **update S with Best_VAL($P_{VAL}(f,g)$)**
15 f_{VAL} =f; g_{VAL} = g
16 **end-if**
17 **end-update S**
18 **Interchange back VAL dB with OPT dB**
19 P(f,g+1) = New_population(P'(f,g), P(f,g))
20 **end for**
21 **F**(f=1) = Best_Indv(P(f)) // Selected_Indv(f=0)
22 **for** *f=1 to Max_Nb_features* //*f: feature*
23 **EFprior**(f-1) = **EF**(f-1) ⊕ **EF**(f-2) ⊕ … ⊕ **EF**(1)
24 g =0
25 Initialize population P(f,g)
26 Evaluate fitness(P(f,g) , **EFprior**(f-1))
27 **$P'_{VAL}(f,g)$ = Apply P'(f,g) interchanging OPT dB with VAL dB**
28 **Evaluate fitness($P'_{VAL}(f,g)$, EFprior**(f-1) **)**
29 **Update Auxiliary archive S:**

```
30        if fit(Best_VAL(P_VAL(f,g)) > fit(Best_Indv_VAL(P_VAL(f-1,g= Nbr_Gen))
31           update S with Best_Indv_VAL(P_VAL(f,g))
33           f_VAL =f; g_VAL = g
34        end-if
35     end-update S
36     Interchange back VAL dB with OPT dB
37     for g =1 to Nbr_Gen
38        Run_Evol (f, g>0)
39        P'_VAL(f,g) = Apply P'(f,g) interchanging OPT dB with VAL dB
40        Evaluate fitness(P'_VAL(f,g), EFprior(f-1))
41        Update Auxiliary archive S:
42          if fit(Best_VAL(P_VAL(f,g)) > fit(Best_Indv_VAL(P_VAL(f-1,g-1)
43             update S with Best_Indv_VAL(P_VAL(f,g))
44             f_VAL =f; g_VAL = g
45          end-if
46        end-update S
47        Interchange back VAL dB with OPT dB
48        P(f,g+1) =  New_population( P'(f,g), P(f,g) )
49     end for
50     Selected_Indv(f) = Best_Indv(P(f))                    // Selected_Indv(f)
51     Bot_stopping(Selected_Indv(f), EFprior(f-1), f ):
52        Def_Sol = EFprior(f-1)= EF(f-1) ⊕ EF(f-2) ⊕ ... ⊕ EF(1)   // no more feat.
53  end-for f=1 to Max_Nb_features
54  Change OPT dB with TEST dB
55  Rr-OPT = Rec_rate{ Def_Sol = EF(f-1) ⊕ EF(f-2) ⊕ ... ⊕ EF(1) }
56  Def_Sol_VAL = Best_Indv_VAL(P_VAL(f_VAL,g_VAL)) ⊕ EF(f_VAL-1) ⊕ ... ⊕ EF(1)
57  Change VAL dB with TEST dB
58  Rr-VAL =Rec_rate { Def_Sol_VAL }
59   Designate results:
    { Best_Indv_VAL(P_VAL(f_VAL,g_VAL)) ⊕ EF(f_VAL-1) ⊕ ... ⊕ EF(1), f_VAL, g_VAL }

end – Bot procedure with Global Validation
```

ANNEX VI

RESULTS EVOLVED CLASSIFIERS OF DIFFERENT REC. RATES

RS	Island	Rec. Rate (%) VAL	Rec. Rate (%) TEST	#*Fave*	Rec. Rate (%) 1-NN
RS02	1	74.14±4.14	87.36±2.01	18.66±2.83	88.87
	2	79.11±0.94	88.81±0.56	26.33±1.72	
	3	76.49±3.76	87.68±1.97	25.20±1.23	
RS04	1	75.72±4.26	87.73±2.59	21.50±1.69	89.28
	2	81.12±2.15	90.05±1.03	23.83±1.54	
	3	80.64±3.45	88.09±1.85	22.13±1.46	
RS07	1	83.04±4.52	88.12±4.62	26.50±2.76	91.40
	2	81.95±6.37	90.36±5.46	20.83±5.33	
	3	84.13±3.28	90.66±1.87	28.13±1.27	
RS14	1	82.67±2.26	88.84±2.60	22.83±1.81	90.99
	2	78.73±9.64	84.60±6.23	28.66±1.98	
	3	83.59±2.33	90.41±2.35	24.83±3.16	
RS28	1	80.01±6.72	87.08±5.59	22.83±2.78	85.71
	2	79.15±8.72	84.42±5.41	22.13±2.26	
	3	81.17±3.49	85.76±2.07	30.00±2.43	
RS29	1	75.42±4.15	83.51±2.02	26.33±2.85	87.85
	2	80.06±2.54	85.18±1.50	22.33±2.24	
	3	79.08±2.04	84.19±1.44	23.66±2.20	
RS62	1	86.72±2.94	92.12±2.28	2.83±2.78	92.48
	2	88.06±1.85	92.23±2.38	22.33±2.88	
	3	87.05±2.15	92.31±1.97	22.53±2.08	
RS71	1	81.27±1.76	87.85±1.10	17.83±3.04	89.88
	2	85.42±1.35	89.60±1.95	20.33±2.15	
	3	86.82±2.33	89.86±2.08	22.33±1.27	
RS86	1	84.36±2.93	89.56±1.38	21.13±2.59	90.17
	2	83.47±4.33	88.43±3.43	25.13±1.15	
	3	82.50±5.04	88.09±3.51	23.83±1.36	

BIBLIOGRAPHY

Alba, Enrique and José M. Troya, "A survey of parallel distributed genetic algorithms", *Complexity*, vol.4, no.4, p.31-52, Mar./Apr. 1999.

Banzhaf, W. and Peter Nordin and Robert E. Keller and Frank D. Francone, *Genetic Programming: An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA. 1998.

Beasley, D. and D. R. Bull, and R. R. Martin, "An Overview of Genetic Algorithms: Part I, Fundamentals". *University Computing*, vol. 15, no. 2, pp. 58-69, 1993.

Benahmed, Nadia. « Optimisation de Réseaux de Neurones pour la Reconnaissance de Chiffres Manuscrits Isolés : Sélection et Pondération des Primitives par Algorithmes Génétiques ». Master's thesis, École de technologie supérieure, 2002.

Bot, Martijn C.J., "Feature Extraction for the k-Nearest Neighbour Classifier with Genetic Programming". EuroGP 2001, *Lecture Notes in Computer Science*, vol 2038, pp. 256-267, 2001.

Burke, E.K. and Gustafson, S. and Kendall, G., "Diversity in genetic programming: an analysis of measures and correlation with fitness," *Evolutionary Computation, IEEE Transactions on*, vol.8, no.1, pp. 47-62, Feb. 2004.

Cantú-Paz, E. and D. E. Goldberg, "Efficient parallel genetic algorithms: Theory and practice," *Computer Methods in Applied Mechanics and Engineering*, 2000.

Cantú–Paz, E., "Parameter setting in parallel Genetic Algorithms", *Studies in Computational Intelligence (SCI)*, 54, 259-276, 2007.

Cunningham, Padraig and John Carney, "Diversity versus Quality in Classification Ensembles Based on Feature Selection", *Machine Learning , 11th European Conference on Machine Learning*, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings ECML 2000: 109-116.

P.A. Devijver and J. Kittler, *Pattern Recognition, a Statistical Approach*, Prentice Hall, Englewood Cliffs, London, 1982.

Dos Santos, E. "PhD Thesis proposal" École de technologie supérieure, Université du Québec, Montréal, Québec, Canada, August, 2004.

Dos Santos, E.M. and Sabourin, R. and Maupin, P., "Single and Multi-Objective Genetic Algorithms for the Selection of Ensemble of Classifiers", *IEEE World Congress on*

Computational Intelligence (WCCI 2006) - International Joint Conference on Neural Networks (IJCNN 2006), Vancouver, BC, July 16-21, 2006.

Duda, Richard and Peter Hart and David Stork, Pattern *Classification*, Second edition 2001.

Fernández de Vega, Francisco and Marco Tomassini and Leonardo Vanneschi, "An Empirical Study of Multipopulation Genetic Programming", *Genetic Programming and Evolvable Machines* 4(1): 21-51, 2003.

Ferri, F. J. and V. Kadirkamanathan, and J. Kittler, "Feature subset search using genetic algorithms", In *Proceedings of the IEE/IEEE Workshop on Natural Algorithms in Signal Processing (NASP 93)*, pages 23/1–23/7, 1993.

Ferri, F. J. and P. Pudil and M. Hatef and J. Kittler, "Comparative study of techniques for large-scale feature selection", In *Pattern Recognition in Practice IV, Multiple Paradigms, Comparative Studies and Hybrid Systems,* eds. E. S. Gelsema and L. S. Kanal. Amsterdam: Elsevier, pp. 403-413, 1994.

Folino, G. and Pizzuti C. and G. Spezzano, "Ensemble Techniques for Parallel Genetic Programming based Classifiers", *EUROGP 2003*, LCNS Springer-Verlag, 2003.

Folino, G. and C. Pizzuti and G. Spezzano and L. Vanneschi and M. Tomassini. "Diversity analysis in cellular and multipopulation genetic programming", In Ruhul Sarker and Robert Reynolds and Hussein Abbass and Kay Chen Tan and Bob McKay and Daryl Essam and Tom Gedeon *editors*, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003,* pages 305-311, Canberra, 2003. IEEE Press.

Gagné, Christian and Marc Parizeau and Marc Dubreuil, "Distributed BEAGLE: An Environment for Parallel and Distributed Evolutionary Computations". In *Proceedings of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS), 2003*, Sherbrooke (QC), 2003.

Gagné, Christian and Parizeau, Marc, 2004a, "Open BEAGLE Manual", *Technical report RT-LVSN-2003-01-V213-R1*, April 13 2004.

Gagné, Christian and Parizeau, Marc, 2004b, "Genericity in Evolutionary Computation Software Tools: Principles and Case-Study", *Technical report RT-LVSN-2004-01*, October 28 2004.

Goldberg, David E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1989.

Guerra-Salcedo, C and L.D. Whitley, (1999a) "Genetic approach to feature selection for ensemble creation," In: *W. Banzhaf, J. Daidam, A.E. Eiben, M.H. Garzon, V.*

Honavar, M. Jakiela, R.E. Smith (Eds.), Proceedings of the 1999 International Conference on Genetic and Evolutionary Computation, (GECCO-99), Orlando, FL, Morgan Kaufnlann, San Mateo, CA, pp. 236-243, 1999.

Guerra-Salcedo, C. and Whitley, D. (1999b), "Feature Selection Mechanisms for Ensemble Creation: a Genetic Search Perspective," In *Data Mining with Evolutionary Algorithms: Research Directions,* AAAI Press, Orlando, Florida, pp 13-17, 1999.

Guo, Hong and L.B. Jack and A.K. Nandi, "Feature generation using genetic programming with application to fault classification", *IEEE Transactions on SMC- Part B: Cybernetics*, Vol 35, No1, February 2005.

Guo, Hong and Asoke K. Nandi, "Breast cancer diagnosis using genetic programming generated feature," In *Pattern Recognition*, Volume 39, Issue 5, , May 2006, Pages 980-987.

Herrera, Francisco and Manuel Lozano and José L. Verdegay, "Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis" In *Artificial Intelligence*, Rev. 12(4): 265-319, 1998.

Hirsh, Haym and Wolfgang Banzhaf and John R. Koza and Conor Ryan and Lee Spector and Christian Jacob "Genetic Programming", In *IEEE Intelligent Systems*, Vol 15 No3, pp74-84, 2000.

Ho, T. K., "Nearest Neighbours in Random Subspaces", *Proceedings of the 2nd Intl. Workshop on Statistical Techniques in Pattern Recognition*, pp. 640-648, Sydney Australia, August 1998b.

Ho, T. K., "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832--844, 1998a.

Holland, John H., *Adaptation in natural and artificial systems*, MIT Press, Cambridge, MA, 1992.

Hong, Jin-Hyuk Hong and Sung-Bae Cho, "The classification of cancer based on DNA micro array data that uses diverse ensemble genetic programming", *Artificial Intelligence In Medicine*, 36(1):43-58, 2006.

Ko, A. and Sabourin, R.and Britto Jr. A. and Oliveira, L., "Pair wise Fusion Matrix for Combining Classifiers", *Pattern Recognition*, vol. 40, 2007, pp. 2198-2210.

Iba, H. "Bagging, Boosting, and Bloating in Genetic Programming", *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, 1999.

Jain, Anil and Douglas Zongker, "Feature Selection: Evaluation, Application, and Small Sample Performance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 153-158, Feb., 1997.

Jain, Anil and Robert P.W. Duin and Jianchang Mao, "Statistical Pattern Recognition: A Review," *IEEE Tran. on Pattern Analysis and Machine Intelligence*, vol 22, No 1, Jan. 2000.

Jabeur, K. and Guitouni, A., "Automated learning multi-criteria classifiers for FLIR ship imagery classification," In *Proceedings of International Conference on Information Fusion*, Québec, CA, 2007.

Koza, John R. *Genetic Programming: On the Programming of Computers by Means* of *Natural Selection*, Cambridge, MA: The MIT Press. 1992.

Koza, John, R., "Introduction to Genetic Programming: Tutorial", In *GECCO-2004 Proceedings of the Genetic and Evolutionary Computation Conference*, Seattle June 27, 2004.

Koza, J. and Keane, M. and Streeter, M. and Mydlowec, W. and Yu, J. and Lanza, G. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Springer, 2005 Second edition.

Krawiec, Krzysztof, "Genetic Programming-based Construction of Features for Machine Learning and Knowledge Discovery Tasks", *Genetic Programming and Evolvable Machines* 3(4): 329-343, 2002.

Kubalik, J. "Evolutionary techniques - Foundations of Artificial Intelligence", On-line 28 p. http://cyber.felk.cvut.cz/gerstner/HUT2000/ga/ga.ppt, consulted on December 19, 2006.

Kudo, M. and Sklansky, J., "Comparison of algorithms that select features for pattern classifiers," In *Pattern Recognition*, v33. p. 25-41.

Langdon, W. B., *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!,* Kluwer Academic Publishers, 1998.

Lin, S. C. and W. F. Punch and E. D. Goodman, "Coarse-grain parallel genetic algorithms: Categorization and a new approach," In *Sixth IEEE SPDP*, pp. 28-37, 1994.

Michalewicz, Z. and Schmidt, M., "Parameter Control in Practice," In *Parameter Setting in Evolutionary Algorithms*, Lima, C., Lobo, F., and Michalewicz, Z. (Editors), Springer Series *Studies in Computational Intelligence*, 2007.

Mitchell, T.M., *Machine Learning*, McGraw Hill Science/Engineering/Math, 1st edition, March 1997.

Mitchell, M., *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, USA, 1999

Muni, Durga Prasad and Nikhil R. Pal and J. Das, "Genetic Programming for Simultaneous feature Selection and Classifier Design," *IEEE Transactions on Systems, Man and Cybernetics-B*, No. 1, Vol. 36, Feb. 2006.

Newman, D.J. and Hettich, S. and Blake, C.L. and Merz, C.J. 1998, "UCI Repository of machine learning databases", on-line http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science. Consulted on Jan. 26th, 2007.

Oliveira, L.S., Sabourin, R., Bortolozzi, F., and Suen, C.Y. "Automatic Recognition of Handwritten Numerical Strings: A Recognition and Verification Strategy," In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, N. 11, Pages 1438-1454, IEEE Computer Society Press, 2002.

Oliveira, L.S., Sabourin, R., Bortolozzi, F. et Suen, C.Y., 2003a, "A Methodology for Feature Selection Using Multi-Objective Genetic Algorithms for Handwritten Digit String Recognition", In *the International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, Vol 17, No. 6, pp 903-929, 2003.

Oliveira, L.S. 2003b "Automatic Recognition of Handwritten Numerical Strings", Doctoral Dissertation, Montreal, École de technologie supérieure, 2003.

Oliveira, L.S., Sabourin, R., Bortolozzi, F. et Suen, C.Y., 2003c, Feature Selection for Ensembles : A Hierarchical Multi-Objective Genetic Algorithm Approach, In *the Proceedings of the 7th International Conference on Document Analysis and Recognition - ICDAR2003*, pp.676-680, Edinburgh, Scotland, 3-6 August 2003.

Opitz, D., "Feature selection for ensembles," In *Proceedings of the Sixteenth National Conference on Artificial Intelligence.* Menlo Park, CA: AAAI Press, 1999, pp. 379—384.

Opitz, D. and R. Maclin, "Popular ensemble methods: An empirical study," In *Journal of Artificial Intelligence Research*, 11:169--198, 1999.

Otero, Fernando E. B. and Monique M. S. Silva and Alex A. Freitas and Julio C. Nievola. "Genetic Programming for Attribute Construction in Data Mining". In *Genetic Programming, Proceedings of EuroGP'2003* Conor Ryan and Terence Soule and

Maarten Keijzer and Edward Tsang and Riccardo Poli and Ernesto Costa *editors*, , volume 2610, pages 384-393, Essex, 2003. Springer-Verlag.

Pal, S. K. and P. P. Wang, *Genetic Algorithms for Pattern Recognition*. CRC Press, Inc. 1996.

Park, Y. and J. Sklansky, "Automated design of linear tree classifiers", *Pattern Recognition*, v.23 n.12, p.1393-1412, 1990.

Paris, Gregory and Denis Robilliard and Cyril Fonlupt, "Exploring Overfitting in Genetic Programming" In *Pierre Liardet and Pierre Collet and Cyril Fonlupt and Evelyne Lutton and Marc Schoenauer* editors, *Evolution Artificielle*, 6th International Conference, volume 2936, pages 267-277, Marseilles, France, 2003.

Radtke, P. and Wong, T. and Sabourin, R., "An Evaluation of Over-Fit Control Strategies for Multi-Objective Evolutionary Optimization," *IEEE World Congress on Computational Intelligence (WCCI 2006) - International Joint Conference on Neural Networks (IJCNN 2006)*, Vancouver, BC, July 16-21, 2006.

Raymer, M. L. and W. F. Punch, E. D. Goodman, and L. A. Kuhn, "Genetic Programming for Improved Data Mining -- Application to the Biochemistry of Protein Interactions", *Genetic Programming 96*, pg 375-381, July 1996.

Rhéaume, F., Jousselme, A-L, Grenier, D., Bossé, E., and Valin, P., "New Initial Basic Probability Assignments for Multiple Classifiers", in *SPIE Aerosense 2002*, Orlando, Florida, April 1-5 2002, Orlando, Florida, pp. 319-328.

Rosca, J. P. "Entropy-driven adaptive representation," In *Rosca, J. P., editor, Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* (NRL TR 95.2, University of Rochester), pages 23—32, 1995.

Robilliard, Denis and Cyril Fonlupt. "Backwarding : An Overfitting Control for Genetic Programming in a Remote Sensing Application". In *Artificial Evolution 5th International Conference, Evolution Artificielle*, EA 2001, P. Collet and C. Fonlupt and J.-K. Hao and E. Lutton and M. Schoenauer *editors,* volume 2310, pages 245-254, Ed. Springer Verlag Creusot, France, 2001.

Sarma , Jayshree and Kenneth De Jong , "Generation Gap Methods," In *The Handbook of Evolutionary Computation*. D. Fogel T. Baeck and Z. Michalewicz editor(s). Pages C2.7:1–C2.7:6. IOP Publishing and Oxford University Press.

Sherrah, Jamie R. ; Bogner, Robert E. and Bouzerdoum, Abdesselam, "The Evolutionary Pre-Processor: Automatic Feature Extraction for Supervised Classification using Genetic Programming", in *Genetic Programming 1997: Proceedings of the Second Annual Conference, 1997*, pp 304-312.

Skolicki, Z., De Jong, K., "The influence of migration sizes and intervals on island models", *Proceedings of Genetic and Evolutionary Computation Conference*, Washington DC, 2005.

Srinivas, M. and Lalit M. Patnaik, "Genetic algorithms: A Survey," *Computer*, v.27 n.6, p.17-26, June 1994.

Tremblay, G., « Optimisation d'Ensembles de Classifieurs non Paramétriques avec Aprentissage par Représentation Partielle de l'Information » Master's thesis, École de technologie supérieure, 2004.

Tremblay, G. and Sabourin, R. and Maupin, P., "Optimizing Nearest Neighbour in Random Subspaces using a Multi-Objective Genetic Algorithm," *17th International Conference on Pattern Recognition (ICPR2004)*, Cambridge, U.K., 23-26 August 2004, pp 208-211.

Tsymbal, Alexey and Mykola Pechenizkiy and Padraig Cunningham, "Diversity in search strategies for ensemble feature selection," *Information Fusion*, Volume 6, Issue 1, Diversity in Multiple Classifier Systems, March 2005, Pages 83-98.

Vafaie, H. and De Jong, K., "Genetic Algorithms as a tool for restructuring feature space representations," *Proceedings of the seventh International Conference on Tools with Artificial Intelligence*, Herdon, VA 1995.

Vafaie, H. "Using Genetic Algorithms for Restructuring Feature-based Representation Spaces," George Mason University, Fairfax, VA., Ph.D. Thesis, Department, School of Information Technology and Engineering, 1997.

Vincent, Jonathan, "Numerical optimisation using genetic algorithms", 2003. On-line http://dec.bournemouth.ac.uk/staff/jvincent/teaching/ec/numerical_optimisation_using_genetic_algorithms.pdf, Consulted on July 19, 2007.

Wagner, S. and Affenzeller, M., "The Allele Meta-Model - Developing a Common Language for Genetic Algorithms," *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, Lecture Notes in Computer Science 3562, pp. 202-211. Springer-Verlag, 2005.

Whitley, D. and S. Rana and R. B. Heckendorn, "The island model genetic algorithm: On separability, population size and convergence," In *Journal of Computing and Information Technology*, 7(1):33-47, 1999.

Zhang, Yifeng and Siddhartha Bhattacharyya, "Genetic programming in classifying large-scale data: an ensemble method," In *Information Sciences*, 163(1-3): 85-101, 2004.