EVOLUTIONARY COMPUTING FOR FEATURE SELECTION AND PREDICTIVE DATA MINING

By

Muhsin Ozdemir

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Major Subject: Engineering Science

Approved by the
Examining Committee:

_____
Jorge Haddock, Thesis Adviser

_____
Mark J. Embrechts, Thesis Adviser

_____
Curt M. Breneman, Member

_____
Nong Shang, Member

Rensselaer Polytechnic Institute
Troy, New York

March 2002
(For Graduation May 2002)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

I wish to express my deep gratitude to my thesis advisers Professors Jorge Haddock and Mark J. Embrechts, whose support, guidance, and encouragement made this work possible. I wish to express my sincere appreciation to Professor Embrecths for inspiring, directing, and motivating this research. It was a great pleasure to work with him. He is not only an excellent teacher and researcher but also an excellent mentor and friend.

I am also indebted to Dr. Curt M. Breneman and his research group who made valuable contributions to this work by supplying datasets and making helpful suggestions and comments throughout the course of work. I also would like to thank Dr. Nong Shang for his advice and comments throughout the course of work.

I would like to thank all the members of the DDASSL research group for their technical support, help, friendship, and collaboration. During the course of my study I have been fortunate to make many exceptional friends. I thank all of them for their help and collaboration.

Last but not the least, I would like to thank my friends - Fabio Arcienages, Jinbo Bi, Jiaqi Hu, Natasha Yakovchuk, Qiong Luo, Yuchun Yang, Robert Bress, Minghu Song, and Michinari Momma - for all their support and friendship.

I am grateful to the Adnan Menderes University, Turkey, and the NSF for financially supporting me during my study at the Rensselaer.

This thesis is dedicated to my parents, whose constant encouragement throughout the years at the Rensselaer has been a source of inspiration.

ABSTRACT

Feature selection has recently been the subject of intensive research in data mining, especially for datasets with a large number of descriptive attributes such as QSAR (Quantitative Activity Structure Relationship) data. QSAR is an in-silico drug design methodology, which requires identifying important features of molecules that explain a drug relevant activity of interest. A typical QSAR dataset for predicting an activity of interest is characterized by a large number of descriptive features (300 - 1000) for a relatively small number of compounds (typically around 50 - 500).

Finding the best feature subset for a given problem with N number of features requires evaluating all 2^N possible subsets. The best feature subset also depends on the predictive modeling, which will be employed to predict the future unknown values of response variables of interest. Feature selection involves minimizing the number of relevant features for maximizing the predictive power of the model. From this point of view feature selection can be viewed as a special type of multi-objective optimization problem.

Evolutionary computing can be applied to problems where traditional methods are hard to apply or lead to unsatisfactory solutions (e.g. local optima). The methods of evolutionary computation are stochastic and their search methods imitate and model some phenomena from nature and evolution: i) the survival of the fittest and ii) genetic inheritance. This dissertation addresses evolutionary algorithms for feature selection and predictive modeling for QSAR data sets.

CHAPTER 1

Introduction

The purpose of this dissertation is to explore evolutionary computing techniques (i.e., genetic algorithms and evolutionary algorithms) and apply them to feature selection problems for predictive data mining. There are many problems in real life for which no algorithm has been developed to solve at optimality within a reasonable time frame. Several of these problems can be framed as multi-objective optimization problems. Optimization is a process to find a possible best solution satisfying two or more objective amongst countable infinite number of candidate solutions. One class of optimization problems is called combinatorial optimization problems, which arise in situations where one has to combine a set of entities in a specific way.

Problems considered in this dissertation are combinatorial problems. A classic combinatorial optimization problem is the Traveling Salesman Problem (TSP) that can be stated as the problem of finding the shortest path, which goes through each city of a given set of cities once and only once and ends in the starting city. Many combinatorial optimization problems like the Traveling Salesman problem can be formulated in the abstract as finding from a set of $\boldsymbol{S}$ a subset $\boldsymbol{T}$ that satisfies desired criteria and optimize (generally minimize) an objective function $\boldsymbol{f}$. Another combinatorial problem, which is the main topic of this dissertation, is feature selection. Feature selection is a common task in many classification and regression problems. Feature selection involves minimizing the number of relevant features for maximizing the predictive power of the model. From

this point of view feature selection can be viewed as a special type of multi-objective optimization problem.

Evolutionary computing can be applied to problems where traditional methods are hard to apply or lead to unsatisfactory solutions (e.g. local optima). The methods of evolutionary computation are stochastic and their search methods imitate and model some phenomena from nature and evolution: i) the survival of the fittest and ii) genetic inheritance. A well-known evolutionary computing algorithm is the Genetic Algorithms (GA), which are very efficient to search large solution spaces and a good alternative method to solve combinatorial problems. In this dissertation, multi-objective evolutionary computing will be applied to the Traveling Salesman Problem and to feature selection problem for predictive data mining.

Because the TSP problem is a well-known problem, evolutionary algorithms will be applied to the TSP for benchmarking purposes and novel evolutionary algorithms will be developed to solve the TSP-like problems (e.g. feature selection problem). These algorithms will then be applied to feature selection in predictive data mining.

Data mining is the process to automate the discovery of non-obvious, novel, and potentially useful information from large datasets. The data mining problems can broadly be divided into two groups: i) predicting of future unknown values of some dependent (response) variables in a data set by using some or all of the other variables (features) in the data set, and ii) describing the data by revealing hidden patterns or information that can be easily interpretable by users or experts. We define a prediction related data-mining problem as the *standard data-mining problem*. In this context, a standard data-mining problem is a multivariate regression or classification problem for which there are many

candidate features to choose from. The purpose is not only to build a good predictive model but also to explain and interpret to some degree how and why the model works. In order to succeed in this task a first step is feature selection. Feature selection is very important because excess features cause a curse of dimensionality problem and make the predictive model more difficult to explain and interpret. Data-strip mining is defined as an iterative procedure for feature selection for data sets where the number of features often exceeds or is on the order of the number of data records. The second step is to build a predictive modeling.

The outline of this dissertation is the following:

Chapter 2 contains a review of evolutionary computation methods for optimization problems.

Chapter 3 is dedicated to genetic algorithms. Genetic algorithms provide an alternative to traditional optimization methods by using powerful search techniques to locate near optimal solutions in complex optimization problems. The representations, selection schemes, genetic operators for the genetic algorithms were presented.

Chapter 4 describes the Traveling Salesman Problem, which is a well-studied classic combinatorial optimization problem in many areas, including evolutionary computation. Genetic algorithms for the TSP with a modified Partially Mapped Crossover operator that was originally proposed by Goldberg and Lingle [1] are further modified for the problem. An algorithm for the TSP based on evolutionary programming algorithm was also developed. The proposed algorithm, Evolutionary Programming with Constant Population (EPC) is different from the standard evolutionary programming. First, EPC always keeps a constant number of individuals similar to genetic algorithms.

Second, EPC uses a selection scheme (i.e., simulated annealing) with a mutation operation.

Chapter 5 presents a literature review for evolutionary algorithms for feature selection, especially for *in-silico* drug design problem. *In-silico* drug design relies on the Quantitative Activity Structure Relationship (QSAR) methodology, first introduced by Hansch et al. [2]. The aim is to predict the biological activity of new untested chemicals from the knowledge of their chemical structures. QSAR methods deal with identifying important features of molecules that are relevant to explain variations in an activity of interest. A typical QSAR dataset for predicting an activity of interest is characterized by a large number of descriptive features (300-1000) for a small number of compounds (molecules).

Chapter 6 introduces data strip mining. Data strip mining is an iterative procedure for feature reduction/model building for datasets where the number of features exceeds or is on the order of the number of data records. The backbone of the strip mining is the sensitivity analysis, which determines the saliency of each of the features in a model and to reduce the number of features for the model [3-5].

Chapter 7 describes three QSAR datasets (Lombardo, HIV_{rt}, and Caco2), which were utilized for benchmarking the algorithms proposed in this dissertation. Since the aim of the QSAR is to predict the biological activity of new untested molecules from the knowledge of their chemical properties, a brief description about the descriptive features is also presented.

In Chapter 8, three evolutionary algorithms for the feature selection problem are proposed: i) genetic algorithm with a floating-point representation, ii) genetic algorithm

with a unique list representation iii) evolutionary programming algorithm with a unique list representation. A novel GA-based feature selection methodology (GAFEAT) based on the correlation matrix is introduced. In GAFEAT, the GA determines which descriptive features have the best correlation with the response but have a relatively weak inter-correlation. GAFEAT is a filter method, which selects features based on the training data alone by using correlation-based evaluation function without taking the biases of modeling method into consideration [6].

In Chapters 9 and 10, GAFEAT feature selection is extended by replacing a correlation-based fitness function with Partial Least Squares (PLS) regression and Local Learning (LL) algorithms in order to take into consideration the biases of the modeling algorithms, respectively. In Chapter 11 some suggestions for future research are given.

CHAPTER 2

Optimization

In a daily life, we often face to situations that involve optimization. For example, such an optimization problem might involve deciding the best route to work or can be complex as to pick a combination of variables that produce a good predictive model in a multivariate analysis [7]. Any abstract task to be accomplished can be thought of as solving a problem, which, in turn, can be perceived as a search through a potential solution space [8]. It is not possible to optimize when there is only one way to carry out a task without any alternatives. As an example, if there is only one way to drive to work, there is no choice but to take that route whether it is short or not. We consider optimization process if a decision between alternatives needs to be made whenever two or more solutions (alternatives) exist and if it is desirable to choose the best one. The Merriam Webster dictionary defines the term of optimization as "*an act, process, or methodology of making something (as a design, system, or decision) as fully perfect, functional, or effective as possible*." The optimization endeavors to improve performance towards some optimal point or points [9]. Independent features that distinguish the alternatives from one another are called (independent) variables or parameters of the system under consideration; they may be represented as a binary (discrete) or integer (real values) depending on the particular problem at hand [10]. Making a rational decision between alternatives requires a value judgment to decide which solution can be classified as the better or the best solution. This value judgment usually relies on evaluating an objective function, which depends on a metric for the system performance and is functionally related to parameters

of the system. An important and often the most difficult step in an optimization process is to define an appropriate objective function for the problem at hand. There may exist multiple objectives at the same time, and the relative weights of each of these corresponding goals should be considered as well [11].

2.1 Optimization Methods

The lack of recognizing a universal optimization method explains the existence of numerous optimization methods, each with limited application to a special case [10]. Figure 2.1 divides optimization methods into three broad classes [9]: calculus-based, enumerative and random search methods.

Calculus-based methods also subdivide into indirect (analytic) and direct (numeric) methods. Indirect methods search local optima by solving the usually nonlinear set of equations resulting from setting the gradient of the objective function equal to zero [9]. They attempt to find the optimum in a single step, without tests or trials [10]. On the other hand, direct methods search local optima by hopping around the search space and assessing the gradient of the new point, which guides the search. This is simply the notion of “hill-climbing”, which finds the best local point by climbing the steepest permissible gradient [9]. Hill-climbing methods approach the solution in a stepwise manner. At each step it is hoped that the value of the objective function will improve [10]. The common characteristic of hill-climbing methods is that they begin with an initial solution (usually determined randomly or deterministically by users) and a new solution is calculated based on the current solution. The algorithms differ from each other according to the update rules used to create a solution.

In general, direct optimization methods are computationally prohibitive, and they tend to work for simple unimodal functions or specialized applications. On the other hand, indirect methods require the calculation of gradients of functions and constraints. Most of the these methods do not guarantee to find the global optimal solutions, because these algorithms usually terminate when the gradient of the objective function is very close to zero, which may occur both in case of local and global solutions. The other obvious drawback of indirect methods is the calculation of gradient, which may be expensive or not well defined. In many real world optimization problems the gradient of the objective function and constraints may not be calculated exactly because the objective function and/or constraints cannot be written in explicit mathematical form [12].

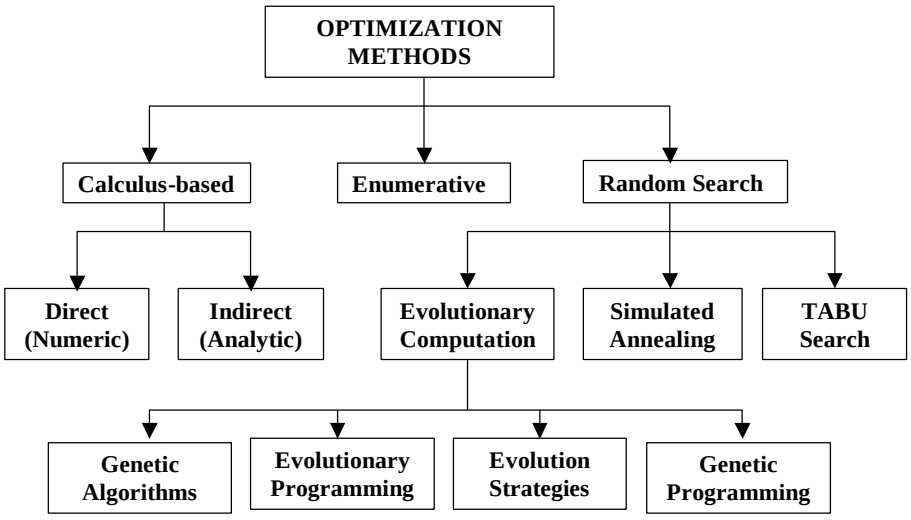


Figure 2.1 Classification of Optimization Methods

Enumerative methods search every point related to an objective function's search space, one point a time. They are very simple to implement but computation times may be prohibitive. Dynamic programming is a good example of this kind of search algorithm.

The shortcomings of the calculus-based and enumerative methods have led to the random search algorithms. It is common to resort to random decisions in optimization whenever deterministic rules do not have the desired success [10]. Randomized search does not necessarily imply arbitrary search. Random search methods are based on enumerative methods but exploit additional information to guide the search. They cannot perform better than enumerative methods in the long run [9]. These methods can be divided into at least three subclasses: simulated annealing, evolutionary computation and TABU search algorithms.

2.2 Simulated Annealing

Simulated Annealing (SA) was first introduced by Kirkpatrik et al. [13] for solving hard combinatorial optimization problems and is different from the biologically motivated evolutionary algorithms [10]. SA works by emulating the annealing phenomenon from material science. Annealing is the physical process of heating up a material until it melts, followed by cooling it down in a controlled way until material crystallizes into a state with perfect lattice. During this process, the free energy of the material is minimized. The cooling process must proceed carefully in order to escape from locally optimal lattice structures with crystal imperfections [14]. This SA process for optimization can be formulated as a problem of finding a solution with minimal cost among the very large number of possible states. The physical annealing process can be modeled by computer simulation methods based on Monte Carlo techniques. SA is based on Monte Carlo techniques and generates a sequence of states of the solid following way. Given a current state *i* of the solid with energy $\mathbf{E_i}$, then next state **j** is generated by

applying a perturbation which transforms the current state into a next state by a small change. The energy of this new state is $\mathbf{E_j}$. If the energy difference, $\mathbf{E_j - E_i}$, is less than or equal to zero, the state **j** becomes the current state. If the energy state is greater than zero, the state **j** is accepted with a certain probability. The acceptance probability is given by

$$P_j\{\text{accept}\} \begin{cases} 1 & \text{if } E_i > E_j \\ \exp\left(\dfrac{E_i - E_j}{k_B T}\right) & \text{if } E_i \le E_j \end{cases},$$

where **T** denotes the temperature of the heat bath and $\mathbf{k_B}$ a physical constant known as the Boltzman constant. The acceptance rule described above is known as the Metropolis criterion and the algorithm that uses this acceptance criterion is known as Metropolis algorithm [14]. If the temperature is lowered slowly, the material can reach thermal equilibrium at each temperature. In the SA algorithm, this is achieved by generating a large number of transitions at a given temperature value. Thermal equilibrium is characterized by the Boltzman distribution [15].

2.3 TABU Search

TABU search (TS), first suggested by Glover [16], is an iterative heuristic procedure for solving discrete combinatorial optimization problems. The basic idea behind the method is to explore the search space of all feasible solutions by a sequence of moves. A move from one solution to another is the best available solution in the neighborhood of the current solution. However, a set of solutions determined by the short-term and the long-term history of the sequence of moves is forbidden (or taboo) in order to escape from locally optimal solutions. TS can be thought as an extension of local/neighborhood search with the inclusion of memory structures that record and

exploit the history of the search in order to escape local optima and lead the search toward higher quality solutions [17]. It has been successfully applied to obtain optimal or near optimal solutions to scheduling problems, the traveling salesman problems, and layout optimization applications.

2.4 Evolutionary Computation

Evolutionary computation incorporates algorithms that are inspired from evolution principles in nature [18]. The methods of evolutionary computation algorithms are stochastic and their search methods imitate and model some natural phenomena: i) the survival of the fittest and ii) genetic inheritance. Evolutionary computing can be applied to problems where traditional methods are hard to apply (e.g. gradients are not available) or lead to unsatisfactory solutions (e.g. local optima) [18]. Evolutionary algorithms work with population of potential solutions (i.e. individuals). Each individual is a potential solution to the problem under consideration and is encoded into a data structure suitable to the problem. Each encoded solution is evaluated by an objective function (environment) in order to measure its fitness. Bias on selecting high-fitness individuals exploits the acquired fitness information. The individuals will change and evolve to form a new population by applying genetic operators. Genetic operators perturb those individuals in order to explore search space. There are two main types of genetic operators: i) mutation and ii) crossover. Mutation type operators are asexual (unary) operators, which create new individuals by a small change in a single individual. On the other hand, crossover type operators are multi-sexual (multary [19]) operators, which create new individuals by combining parts from two or more individuals. After a number

of generations have evolved the process is terminated based on a termination criterion. The best individual in the final is then proposed as a (hopefully near-optimal or optimal) solution for the problem. Evolutionary computing further subdivides into four classes: i) genetic algorithms, ii) evolutionary programming, iii) evolution strategies, and iv) genetic programming. Although there are many close similarities between these evolutionary computing paradigms, there are also profound differences between them [8]. These differences generally concern the level in the hierarchy of evolution being modeled: the chromosome, the individual, or the species [18]. There are also many hybrid methods that combine various features from two or more of the methods described in this section.

2.4.1 Genetic Algorithms

Genetic Algorithms (GAs) are part of a collection of stochastic optimization algorithms inspired by natural genetics and the theory of biological evolution [20]. The idea behind genetic algorithms is to simulate natural evolution to optimize a particular objective function. In the last three decades, GAs have emerged as practical, robust optimization and search methods [8]. In the literature, Holland's genetic algorithm is called Simple Genetic Algorithm (SGA) [21]. SGA works with a population of individuals (chromosomes), which are encoded as binary strings (genes). A detailed literature review about genetics algorithms is presented in Chapter 3.

2.4.2 Evolutionary Programming

Evolutionary programming (EP) was developed by Lawrence Fogel in the late 1960s. Although EP techniques originally aimed at evolving artificial intelligence in the

sense of developing the ability to predict changes in the environment [22, 23], it is often used as an optimizer. After initializing a population of ***N*** individuals and generating ***N*** children by mutation, ***N*** survivors are selected from the population of parent and children using a probabilistic function based on fitness. As a consequence, individuals with a greater fitness have a higher chance to survive to the next generation.

2.4.3 Evolution Strategies

Evolution strategies (ES) are algorithms that mimic the principles of natural evolution as a method to solve parameter optimization problem [8]. They were developed and used in Germany at almost same time that genetic algorithms emerged in the U.S.A. during the late 1960s [10]. Early evolution strategies were based on a population consisting of one individual and one genetic operator (mutation) only. This method called the two-member evaluation strategy. However, the most significant difference between simple genetic algorithms and evolution strategies is the representation of the variables. In ES, an individual was represented as a pair of floating point-valued vectors, i.e., **v = (x,** s**)**. Here, the first vector **x** represents a point in the solution space; the second vector s represents the corresponding standard deviation. The individual is altered by mutation by $x_{t+1} = x_t + N(0, s)$, where $N(0, s)$ is a vector of independent identically normally distributed random numbers with a mean zero and standard devition s. This is consistent with biological observation that smaller changes occur more often than larger ones [8]. The mutated individual replaces its parent if and only if the mutated individual has a better fitness and all existing constraints are satisfied. Otherwise, the offspring vanishes and the population remains unchanged.

Multi-member evolution strategies were also introduced [10]. Here a population consisting of m individuals at generation g produces l offspring. The m best of $(m + l)$ individuals will survive as parents for the next generation. This model allows the more fit individuals to live for a very long time; therefore, it causes the premature convergence. In order to prevent the premature convergence, the following modification is implemented to the multi-member evolution strategy. Here, a population consisting of m individuals at generation g produces l offspring, where $l > m$. The m best of the l offspring are chosen to be parents of the following generation.

2.4.4 Genetic Programming

Genetic programming (GP) was developed by Koza [24]. Koza views many different problems in artificial intelligence, symbolic processing, and machine learning as requiring discovery of a computer program that produces some desired output for particular inputs. The process of solving these problems becomes equivalent to searching a space of possible computer programs for a most fit individual computer programs. Koza [24] developed genetic programming, which provides a way to search for a most fit program for the problem. Genetic programming applies genetic algorithms to a population of computer programs. In order to create new programs from two parent programs, the programs are written as trees. Removing branches from one tree and inserting them into another creates new programs. This process ensures that the new program is also a valid program. Individual programs are evaluated by a fitness function and best solutions selected for modification and re-evaluation. This process is repeated until a most fit program is produced.

2.4.5 Summary for Evolutionary Computing

Evolutionary computing can offer several advantages for solving difficult real-world optimization problems. These advantages related to [25]:

- Conceptual simplicity
- Broad applicability
- Higher performance than classic methods on real problems
- Easily hybridized with other methods
- Suitability for parallel processing (computing)
- Adaptive solutions to changing circumstances
- Capability to optimize its exogenous parameters
- No gradient information necessary

Figure 2.2 [9] shows a generic effectiveness index across a problem continuum for a calculus-based, an enumerative, a random search and an idealized robust scheme. Calculus-based methods perform very well in a narrow problem domain (unimodal) but it is very inefficient elsewhere. As it is expected, enumerative scheme performs low efficiency across the all problem domain. Because random search methods are based on enumerative techniques, they are expected to do no better than enumerative methods.

The "No-Free-Lunch" (NFL) theorem [26] shows that all algorithms that search for optima of a cost function perform exactly the same, when averaged over all possible cost function. As a result, there cannot exist a single algorithm for solving all optimization problems that is consistently better than any other algorithm. The question of whether evolutionary algorithms are inferior or superior to other optimization method

does not make sense. It could be claimed that evolutionary algorithms behave better than other approaches with respect to solving a specific class of problems [27]. As we can see in Figure 2.2, NFL can be justified in the case of evolutionary algorithms versus many classical optimization methods mentioned above. Many classical methods are more efficient in solving linear, quadratic, convex, unimodal, separable, and many other special problems. On the other hand, evolutionary algorithms are more efficient in solving discontinuous, nondifferentiable, multimodal, noisy problems [27].

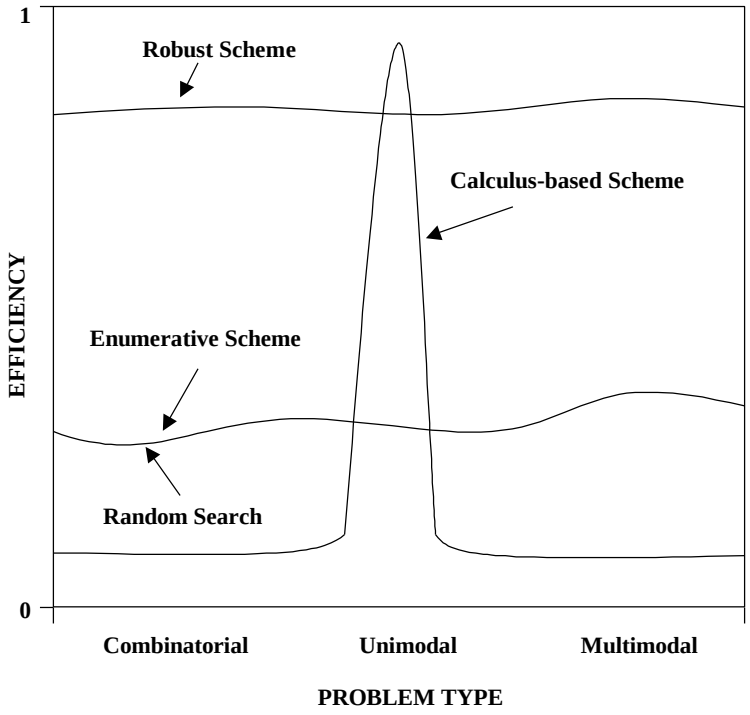


Figure 2.2 Efficiency of Optimization Methods [9]

Since the range and type of the problems faced with in real life are so diverse, it is difficult to develop one good method whose performance curve would be like the Robust Scheme shown in Figure 2.2. An algorithm can be thought of as a robust algorithm that

can successfully solve a wide variety of problems without making too much change in the algorithm. Genetic Algorithm and Simulated Annealing are both robust search and optimization algorithms that apply to a wide variety of search and optimization problems in general [9].

2.5 Combinatorial Optimization

Combinatorial optimization problems arise in situations where one has to combine a set of entities in a specific way. If the quality of the resulting combination of entities can be measured, then combinatorial optimization is the task of finding the best combination of entities. Many combinatorial optimization problems like the Traveling Salesman problem can be formulated in the abstract as finding from a set of $\boldsymbol{S}$ a subset $\boldsymbol{T}$ that satisfies desired criteria and optimize (generally minimize) an objective function $\boldsymbol{f}$ [28]. Most of the combinatorial optimization problems are NP-complete for which it is not guaranteed that an optimal solution can be found even when using the most efficient computers. The computation time required for processing a typical NP problem with input data of size **n** by a "brute force" algorithm requires at least $\mathbf{2^n}$ steps if all subsets of the given **n**-point set of inputs are considered (for example variable selection problem), or on the order of **n!** steps if all permutations are considered (for example, the Traveling Salesman Problem), or $\mathbf{n^n}$ steps if all self-maps are considered [29]. It is believed that optimal solutions cannot be found for NP-hard problems within polynomially bounded computation times [14]. Therefore, solving NP-hard problems at optimality requires unfeasible amount of computation time. This situation led to look for heuristic methods that find a near optimal solution relatively fast within a reasonable time [30].

One of the objectives of this dissertation is to develop a solution methodology using evolutionary computing for combinatorial optimization problems that are similar to the Traveling Salesman Problem (TSP). A detailed literature for relevant TSP like problems is included in Chapter 4.

The Traveling Salesman Problem is a classic combinatorial optimization problem and can be stated as the problem of finding the shortest closed tour, which visits each city of a given set of cities once and only once. Our objective is to find an ordering of $\boldsymbol{N}$ cities that minimize the tour length [30]. Since the TSP is NP-hard [29] it has been attacked by many heuristics methods such as local optimization [28], simulated annealing [13], neural networks [31]. Because finding optimal solution for the TSP involves searching in a solution space that grows exponentially with number of cities, the TSP also has been attacked to solve by genetic algorithms [1]. These algorithms produce near-optimal solutions by maintaining a population of candidate solutions, which evolves by applying crossover and mutation operators with a selection scheme that is biased towards selecting more fit individual [8].

CHAPTER 3

Genetic Algorithms

Genetic Algorithms (GAs) were proposed by John Holland and his students at the University of Michigan in the early 1970s [32] and provide an alternative to traditional optimization methods by using powerful search techniques to locate near optimal solutions in complex optimization problems. GAs are stochastic algorithms whose search methods model some natural phenomena based on genetic inheritance and natural selection. It is a multi-directional search by maintaining a population of potential solutions and assures information formation and exchange between these directions [8]. The potential solutions to a problem evolve to a better-fit group of solutions [20]. At each generation the better solutions reproduce, while the relatively bad solutions eventually die off. GAs have been successfully applied to real world optimization problems like scheduling processes, the traveling salesman like problems, variable reduction, facility layout problems, optimization problems in general. GAs have the following distinct properties:

- Work with encoding of the parameters,
- Search by means of a population of potential solutions,
- Use an evaluation (fitness) function that does not require the calculation of derivatives,
- Search stochastically.

A generic pseudo-code for a genetic algorithm is shown in Figure 3.1.

```
procedure genetic algorithm
begin
        Choose a coding to represent variables
        t ← 0
        Initialize population P(t)
        Evaluate population P(t)

        while (not termination condition) do
                t ← t+1
                Select P(t) from P(t-1)
                Alter P(t) with crossover and mutation
                Evaluate P(t)
        end
end
```

Figure 3.1 A Simple Genetic Algorithm Cycle

A genetic algorithm for a particular problem contains the following components:

- A genetic representation for parameters in the problem,
- A way to create initial population,
- An evaluation (fitness) function,
- Genetic operators (crossover, mutation) that alter the population,
- Values for parameters that genetic algorithm uses (population size, number of generation, probabilities of applying genetic operators, selective pressure, etc.).

These are explained in detail in following sections.

3.1 Genetic Vocabulary

Since genetic algorithms are rooted in natural genetics, most of the nomenclatures in the field are taken from biology. A brief introduction to vocabulary is presented first. In a biological organism, the information specifying how the organism is to be constructed is carried in ***chromosomes***. Organisms whose chromosomes are arrayed in

pairs are called ***diploid*** (see section 3.2.3); organisms with unpaired chromosomes are called ***haploid***. In nature, most of the sexually reproducing species have diploid chromosomes. For example, human being has 23 pairs of chromosomes in each cell [33]. Each chromosome consists of a number of units called ***genes***. The position of a gene on a chromosome is called a ***locus***. The locus of the gene within the chromosome structure determines what particular characteristic the gene represents. A set of values that a gene may take at a particular location on a chromosome is called as ***allele***. One or more chromosomes may specify the complete organism. However, we consider only one-stranded chromosome genotype. The complete set of chromosomes is called a ***genotype***, and the interaction of a genotype with its environment to form an organism is called a ***phenotype***. Therefore, different genotypes may lead to same phenotype, or same genotypes may result in different phenotypes.

3.2 Genetic Representation of Parameters

In a genetic algorithm, the environment is the problem under consideration and each of the organisms is a solution to the problem. Two things must be determined in order to apply a genetic algorithm to a given problem: i) a genetic code representation and ii) a fitness or objective function, which assigns a quality measure to each solution according to its performance [19]. The encoding of the parameters in genetic algorithms depends on the problems at hand.

Around 1972 GA practitioners could be divided into tow camps [9]: i) the minimalist camp and ii) the maximal alphabet camp. The minimalist practitioners have followed Holland's theory of schemata and low-cardinality alphabets. The low-

cardinality-alphabet theory suggests that small alphabets are good, because they maximize the number of schemata available for genetic processing [34]. On the other hand the maximal alphabet practitioners have preferred to represent one parameter with one gene regardless of the number of alternative alleles required for a particular gene. Eventually the two camps for allele representation evolved to binary and floating-point GAs.

3.2.1 Binary Representation and Gray Coding

The traditional method of applying genetic algorithms to real-parameter problems is to encode each parameter as a bit string using either a standard Boolean or a Gray coding [22]. Holland [32] suggested that binary strings should be used for representation of all solutions. The motivation for using binary strings came from the ***schemata theorem*** that was originally introduced by Holland [32]. A ***schema*** is a similarity template that defines a subset of strings with fixed equal genes at certain locations. A schema is therefore a subset of the complete search space. In order to illustrate the concept of schema a special symbol '#'is appended to the binary alphabet. The symbol # is called '*don't care*' symbol that can take any value of alphabet. For instance, the schema (#111) matches two strings, namely {(0111), (1111)} and similarly, the schema (#111#) describes four strings, namely, {(01110), (11110), (01111), (11111)}. A schema represents 2^t strings that have same gene in all positions other than '#', where ***t*** is the number of symbols '#' in the schema template.

Schemata are classified by two parameters: *order* and *defining length* of the schema. A schema's order defines the number of fixed alleles in the chromosome

(string). Its defining length is the distance between the first and last defined allele, ranging from zero up to ***l***-1, where ***l*** is the length of the chromosome. These parameters are useful for determining how likely a schema is to survive after crossover and mutation operators. The order strongly correlated to how likely the schema is to be destroyed after mutation. More alleles in the schema imply a higher chance that mutation will destroy the schema. The defining length tells how likely the schema is to be destroyed by crossover. Longer schema is more likely destroyed by crossover. If a schema tends to occur in above average fitness strings in the population then, the strings incorporating that particular schema are selected more frequently. Strings are generally destroyed by recombination but short and low-order tend to survive. The schemata theorem says that the schemata with above average fitness value will reproduce in increasing numbers in successive generations, while the schemata with below average fitness values will eventually die off.

The binary representation has some drawbacks when applied to multidimensional, high-precision numerical problem [8]. If the optimization problem is defined over a continuous domain, its real-valued parameters can only have approximate genotype representations because of the finite length of the binary encoding. The desired degree of precision determines the appropriate length for the binary encoding [22]. If the number of parameters is large, the size of the chromosome grows quickly. This, in turn, generates very large search space and reduces the performance of genetic algorithms. The genetic algorithms invest a lot of computational effort in evaluating the least significant digits of the gene. However, the optimal value for these digits depends on the more significant ones [35].

Schraudolph et al. [35] propose a dynamic parameter encoding method as a mechanism that obtains high-precision results and avoids problems mentioned above. In this method at first, a very crude precision binary encoding is applied and the GA is allowed to converge. At this point the resulting population is re-encoded with a high precision binary encoding, and the GA is restarted.

If we assume an integer domain $\mathbf{0}$ to $\mathbf{2^L - 1}$ for an arbitrary function, any point in the domain can be represented using an $\boldsymbol{L}$ bit string. A Gray code is defined as a binary encoding schema that guarantees that points that are next to each other in the integer domain have a Hamming distance of one. Gray coding is better encoding for a standard binary representation for problems with limited degree of non-linearity and a locally correlated structure. The Gray coded representations also induce fewer minima than the corresponding binary representations for these kinds of problems [36-38]. Gray coding forces two points that are close to each other in the representation space to be close in the problem space, and vice versa. This is not always the case with the standard binary representation [8].

3.2.2 Real-coded Genetic Algorithms

Real-coded genetic algorithms use floating-point or other high cardinality encoding in their chromosomes [34]. Real-coded (non-binary) representations are more natural for the specific problems. For instance, in a numerical optimization on continues domain a chromosome is represented as a vector of floating point numbers in which each number corresponds to a variable in the problem [10, 12]. Wright [39] suggests a genetic algorithm that uses real parameter vectors as chromosomes, real parameters as genes, and

real numbers as alleles to optimize problems over several real parameters. Vectors of integers are used for scheduling and ordering problems such as the Traveling Salesman Problem [40].

Real-coded GAs usually adopt mutation operators that perturb the current solution around the current value and are well-suited for hill-climbing in the decision space under the consideration. In these kinds of situations, binary coded genes can easily become stuck on Hamming cliffs. Under the assumptions of a fixed population size, a fixed number of search alternatives, and serial processing of individual loci, it can be shown theoretically and empirically that higher cardinality alphabets converge to a solution more quickly than those coded over a smaller alphabet [34].

3.2.3 Diploidicity

So far, the simplest (haploid) genotype found in nature has been considered. In this model, a single-stranded (haploid) string carries all the information related to the problem under consideration. Although there are many haploid organisms in the nature, most of them tend to be of a relatively uncomplicated life form [9]. More complex organisms tend to have more complex chromosome structures employing diploid or double-stranded chromosomes. In this structure a genotype carries one or more pairs of chromosomes, each of them containing information for the same function. When the pair of genes decode to different function values, the mechanism for eliminating this conflict of redundancy is governed by a genetic operator called dominance. To illustrate this operation, let’s consider a diploid chromosome structure where different letters stand for different alleles.

A b C d E f

A b c D e F

Each letter represents one allele for that position; two alleles from each chromosome compete for that position in order to be expressed in the phenotype. One allele (DOMINANT) takes precedence over the other allele (recessive) at that position. An allele is dominant if it shows up in the phenotype when paired with some other allele. In our demonstration, if we assume that all capital letters are dominant and that all lowercase letters are recessive, the phenotype will be:

A b C d E f

® **A b C D E F**

A b c D e F

At each location, the dominant gene is always expressed but recessive genes are expressed only if both alleles are recessive. The mechanism of dominance can be thought of as a genotype-to-phenotype or genotype reduction mapping [9].

3.3 Crossover

Crossover is the key operator, which makes GAs converge to a good solution. The idea behind crossover is that two or more individuals with high fitness values will create one or more new offspring (children) who inherit the best features of their parents. Since the best features are not known a priori, individuals are recombined randomly under a mechanism of selection, which is biased to more fit individuals. Crossover treats good features as building blocks scattered throughout the population and tries to recombine them into new and improved individuals [41]. Crossover will create worse individuals as

well but they will eventually die off. The crossover operator can be divided into two groups: two-parent (classical crossover) and multi-parent.

3.3.1 Two-parent Crossover

Holland [32] originally proposed one-point crossover. It is a reproduction operator that takes two parent chromosomes and randomly chooses a location on the chromosomes. Figure 3.2 illustrates the one-point crossover for binary representation where the vertical line represents the crossover point.

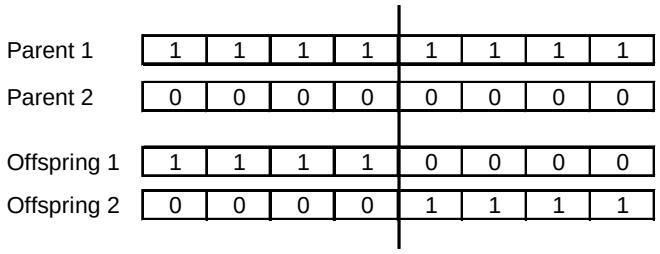


Figure 3.2 One-point Crossover

Chromosomes are cut at that position and the right part of the chromosomes are swapped. Another well-known crossover is two-point crossover that chooses two cut points at random and swaps the middle part of the chromosomes. This can be generalized ***n*** point crossover as well. Figure 3.3 depicts the two-point crossover.

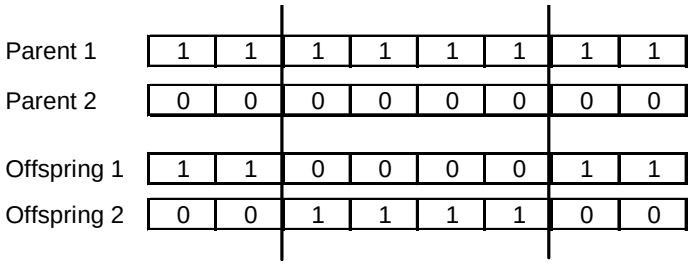


Figure 3.3 Two-point Crossover

3.3.2 Multi-parent Crossover

The creation of new individuals always occurs through either asexual (one parent) or sexual (two-parent) reproduction in nature. Although there are no biological analogies of recombination mechanism where more than two parent genotypes mixed in one single recombination act, there is no necessity to restrict reproduction mechanisms to one (mutation) or two (crossover) parent chromosomes in computer evolution. It has been hypothesized that recombination has a statistical error correction effect, called genetic repair, and that this effect can be improved by using more than two parents for creating offspring [42].

Biasing the recombination operator can improve the performance of the Genetic Algorithm [43]. A problem independent way of incorporating bias into the recombination operation is to use ***n*** parents and apply some limited statistical analysis on allele distribution of the selected parents. Randomly choosing an allele from the parents introduces a very slight bias. Looking at the number of occurrences of a certain allele at a particular position and choosing the most common one introduces a strong bias. Inheriting the number of genes proportional to the fitness value of the parents is a more sophisticated, but still problem independent mechanism.

Eiben [42, 43] has described a number of genetic operators based on multi-parent. These operators use two or more parents to generate a single offspring are based on gene scanning. These operators are uniform scanning, occurrence based scanning, and fitness based scanning.

Uniform scanning is an extension of the number of parents for the uniform crossover operator. Uniform crossover takes two parents and chooses a crossover point

randomly, then, generates two offspring. In uniform scanning, only one offspring is generated and each allele in this single offspring is chosen by a uniform random mechanism, in where each parent has an equal chance of being chosen to provide value. The classical uniform crossover is a very disruptive operator. Applying an n-ary version reduces the level of disruption by using a bigger sample of search space and by creating only one offspring [44].

Occurrence based scanning is based on the value, which occurs in the selected parents in a particular position. The selection of parents is based on their fitness. The selection of value among the marked values is done by a majority vote. If there is no value in majority among the marked values, the value is assigned to the child either a policy (from the first parent, etc.) or randomly. Figure 3.4 illustrates the method.

Parent 1	1	0	1	1	0	0	1	1	1	1	1	0	0	1	1	0
Parent 2	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0
Parent 3	0	1	0	1	1	1	1	0	0	0	0	0	0	0	1	1
Parent 4	0	1	0	1	0	0	1	1	0	1	1	0	1	1	0	0
Child	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**

Figure 3.4 Occurrence Based Scanning on Bit Patterns

Fitness based scanning passes a value to the child proportional to the fitness values of the parents. For instance, for maximization problem, the probability of choosing a value from parent ***i*** is $P_i = \frac{f_i}{\sum f_i}$ (the Roulette wheel selection). E_i, the expected number of genes inherited from parent ***i,*** will be $E_i = P_i * L$, where L is the length of a chromosome.

Adjacency based crossover [43] is a customized case of scanning, which is specifically designed for order-based representations where the relative positioning of values is important. The Traveling Salesman Problem is a good example for these kinds of problems. In this method, the first gene value in the child is always inherited from the first gene value of the first parent. For each parent, its marker is set to the first successor of the previously selected value, which does not already show up in the child. When all immediate successors to a value have already been inherited by the child, the successor of these immediate successors, and so on, will be checked. The new value will be chosen amongst these values that are not present in the child. Figure 3.5 illustrates the method. P(1, 2, 3) and Ch represent parents and child, respectively.

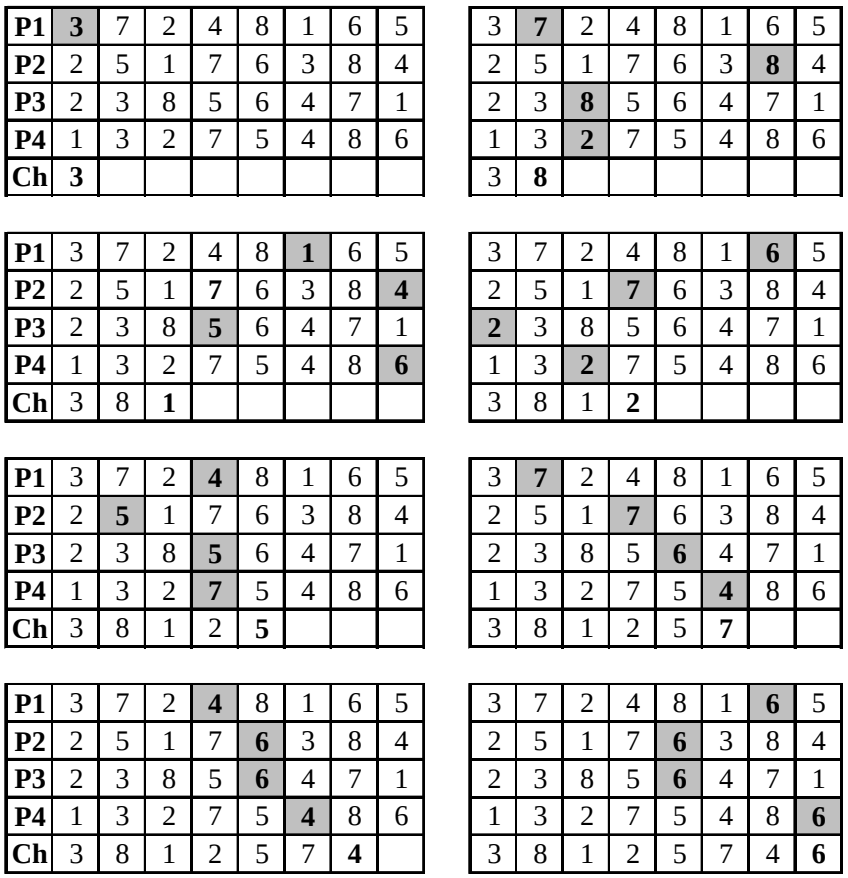


Figure 3.5 Adjacency Based Crossover

Diagonal Multi-parent Crossover has been introduced in [43]. The Diagonal crossover is a generalization of 1-point crossover for two parents generating two offspring. Diagonal crossover uses ***n*** parents (***n*** ³ 2), and (***n*-1**) crossover points and produces ***n*** offspring. Figure 3.6 illustrates the procedure where ***n*=3** and ¯ represents the crossover points.

Parent$_1$	**111111**¯11**11111**¯1111	**Offspring$_1$**	**111111**¯2222222¯3333
Parent$_2$	**222222**¯2222222¯2222	**Offspring$_2$**	**222222**¯3333333¯1111
Parent$_3$	**333333**¯3333333¯3333	**Offspring$_3$**	**333333**¯11**11111**¯2222

Figure 3.6 Diagonal Crossover

The use of more parents in diagonal crossover leads to an improvement in the performance of GA because the search becomes more explorative, without hindering exploitation. The more explorative character is the result of having more crossover points and thus a higher level of disruptiveness, and the fact that using more parents there is more "consensus" to focus on to search a certain region [44]. In order to investigate the effect of having more parents in diagonal crossover, Eiben [45] studied a test suit containing eight numerical optimization problems and established that higher number of parents tented to lead to a better performance.

Lis [46] proposed Multi-Sexual Genetic Algorithm (MSGA) for multi-objective optimization problems. Many real world problems have multiple objectives and these objectives need to be achieved simultaneously. In most cases, these objectives are in conflict with one another, so that it is not possible to improve on any of the objective functions without deteriorating one of the other. This fact is known as the Pareto

optimality concept [47]. A general multi-objective optimization problem consists of a number of objectives and constraints in the form of equalities and inequalities. In these kinds of problems, there may not exist an unambiguous optimal solution. The characteristic of the multi-objective optimization problems is the presence of a large set of acceptable solutions that are superior to the rest of the solutions in the search space when all objectives are considered. On the other hand, these solutions are not optimal from the point of any single objective. MSGA provides each individual with an additional feature (sex or gender) and maps optimization criteria to sexes by one-to-one mapping and evaluates individuals by the optimization criteria belonging to the sex, and uses multi-parent crossover for recombination that requires one parent from each sex. In the recombination phase, the probability of choosing an individual is related to its rank calculated in the evaluation step. The uniform scanning crossover operator creates one child from many parents. The sex of offspring is inherited from the parent supplied the largest number of genes. If there is tie, the parent is chosen randomly. Enforcing that crossover is applied to representatives of different sexes and the different sexes are evaluated by different optimization criteria prevent the algorithm from converging to optimal points with respect to only on single optimization criterion.

3.4 Mutation

Mutation is a common reproduction operator used for finding new points in the search space. When a string (chromosome) is chosen for mutation, a random choice is made for selecting genes of the string, and these genes are modified accordingly. In the case of binary representation, the corresponding bits are flipped form 0 to 1 or vice versa.

A commonly used mutation probability is one over the number of genes in the string [41]. Mutation operator is generally used with a crossover operator as a background operator that diversifies the population [8].

3.5 Selection Mechanism

Selection is one of the main operators used in evolutionary algorithms and its primary objective is to emphasize better solutions in a population [48]. A genetic algorithm starts with a certain number of random (or non random) problem specific structures (population). Each individual or structure in the population represents a solution to the problem and has a fitness value f_i. The GA proceeds for a certain number of iterations (generations) until one or more stopping criteria are satisfied. At the beginning of the each generation, the genetic algorithm performs selection followed by genetic operators (crossover and mutation). There are two important factors in the evolution process of the genetic search [8]: population diversity and selective pressure. These two factors are strongly related with each other because an increase in the selective pressure decreases the diversity of the population, and vice versa. It is very important to balance these two factors in a genetic search because strong selection pressure causes premature convergence and weak selection can make the genetic search inefficient.

3.5.1 Fitness Proportional Selection Schemes

The most common selection mechanism is roulette wheel selection (stochastic sampling with replacement) [9]. Here, each slot on the roulette wheel represents an individual. The area of the slot is directly proportional to the objective function value

(fitness) of the individual. Each individual is selected by a spin of the roulette wheel. Figure 3.7 illustrates the roulette wheel selection scheme.

Individuals	Fitness	Proportional Fitness
Parent1	30	30%
Parent2	60	60%
Parent3	10	10%
Total	100	1

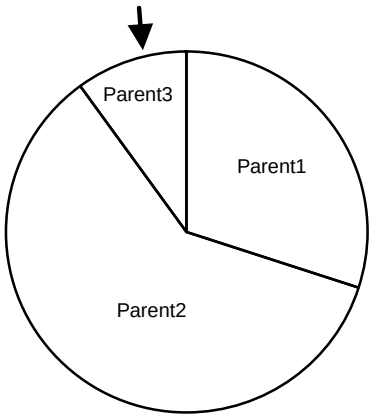


Figure 3.7 Roulette Wheel Selection

Although the lowest fit individual (Parent 3) in Figure 3.7 is selected by the roulette wheel, the individuals with higher fitness are likely to be selected more often the ones with lower fitness. This reproduction method often forces the genetic search to focus on the top individuals only, which leads to premature convergence (local optima).

In order to alleviate the premature convergence caused by the roulette wheel selection, six selection methods were introduced by Brindle [49]. These methods mainly control the number of individual copied to the new population. One of those methods is the *Remainder Stochastic Sampling with Replacement*. This method selects individuals deterministically for the integer part of their expectations and uses the fractional part as a probability for the roulette wheel selection. For example, if the objective function value

of an individual is f_i, the probability of selection of an individual will be $\frac{f_i}{\sum f_i}$. The expected number of individual for each string is $N\frac{f_i}{\sum f_i}$, where ***N*** is population size. For example, an individual with an expected number of copies equal to 2.25, the two copies of the individual are copied to the next population and the third copy will be determined by the roulette wheel with probability 0.25.

3.5.2 Tournament Selection Scheme

Another popular selection method is *Stochastic Tournament Selection* [49]. In this method, ***N*** tournaments are held and each of them contributes one individual to the new population for the next generation. In each tournament two individuals from the current population are selected using roulette wheel selection. The individual with the best overall fitness is selected from them. This process is repeated until the entire population has been replaced. This method was extended to ***k*** number of individuals in each tournament by Goldberg [50]. This method selects some number of individuals and copies the best one amongst them to the new population. If the ***k*** is large, the selective pressure will increase.

3.5.3 Elitist Scheme

The elitist selection scheme was proposed by De Jong [51]. This method copies a certain number of the best individuals from the existing population to the next population. This enforces preserving the best structures for the problem at hand.

3.5.4 Rank Selection

A nonparametric procedure for selection (*Rank Selection*) was introduced by Baker [52]. In this method individuals in the population are sorted according to their fitness values and individuals for the next generation are selected proportionally to their rank rather than actual objective function value. Ranking acts as function transformation that assigns a new fitness value to an individual based on its performance relative to other individuals [53]. Baker [52] used the rank selection to slow down the search speed. This method prevents the super individuals to take over population in a few generations by adjusting the selective pressure. Whitley [53] suggested that selection according to rank is superior to fitness proportionate selection and can also be used to increase search speed. There are linear and nonlinear methods to assign a number of offspring based on ranking [8]. The rank selection has been criticized because it violates the Schema Theorem and ignores information about the search space as revealed by the objective function [53]. On the other hand, rank selection prevent scaling problem and control the selective pressure.

CHAPTER 4

Evolutionary Algorithms for the Traveling Salesman Problem

4.1 Introduction

The Traveling Salesman Problem (TSP) is a classic combinatorial optimization problem and can be stated as the problem of finding the shortest closed tour, which visits each city of a given set of cities once and only once. Our objective is to find an ordering of ***N*** cities that minimize the tour length [30]. Assume that there is a set of cities, S={c_{a1}, c_{b2}, c_{c3}, …, c_{nN}}, where first subscript represents the city name and second subscript represents the visit order of the city. For example, c_{b2} means that city b is visited as a second city after city a. If d_{ij} represents the distance between two consecutively visited cities (for example, d_{23} will be the distance between city b and city c) where i = j+1, 1 ≤ i, and j ≤ N, our objective is to find an ordering of N cities which gives the minimum total tour length. The objective function can be written as

$$\min \left\{ \sum_{i=1}^{N-1} d_{ij} + d_{N1} \right\} \qquad (4.1)$$

We will concentrate in this dissertation on symmetric TSP, in which the distances from city ***i*** to ***j*** is the same as that from city ***j*** to city ***i***. In this case, the tour length does not change if the order of cities are reversed. The distances between cities can in principle be expressed in different metrics, but they must satisfy the triangle inequality. This means that the direct route between two cities is always the shortest path. These types of TSP problems are also called Euclidean Traveling Salesman Problems. Even though, in the

real world, there are cases where the shortest route from city ***i*** to city ***j*** must pass through city ***k***, we will ignore these situations here. It can easily be proven that the optimal solution to the Euclidean TSP can never cross itself [54].

The TSP is very simple to state but difficult to solve exactly. Several variations of the TSP find immediate applications in routing and scheduling problems. In an instance of a TSP, there is a set of ***N*** cities, and a distance between each pair of cities. In order to find a guaranteed optimal solution, which minimizes the tour distance, all the valid tours for a given problem have to be evaluated. For instance, in a 10-city TSP there are $\frac{(N-1)!}{2} = \frac{(10-1)!}{2} = 181{,}440$ distinct tours. If the number of cities becomes 14; then, number of distinct tours becomes 31,135,000,000,000. All of the known algorithms demand computing times that grows exponentially with ***N***. The TSP belongs to the class of NP-hard problems, and consequently, it is unlikely that there is a polynomial-time algorithm that solves each instance of the problem at optimality [31].

4.2 Approaches for Solving the TSP

There are several approaches for solving the TSP. Because the TSP is a NP-hard problem, finding optimal solution requires possibly infeasible computing time. Aarts and Stehouwer grouped these approaches into two categories: optimization and approximation [31]. There are optimization algorithms for the TSP based on enumeration methods using branch and bound techniques that can handle a certain problem size and might take some time to find the optimal solution. In real life there are many situations where much larger sizes of TSP like problem need to be handled (e.g., printed circuit board design).

Several applications stimulated the emergence of approximate heuristic algorithms that can find near-optimal solutions preferably with small running times. Although these heuristic techniques are relatively faster than strict optimization algorithms, their solution qualities are moderate. Approximation algorithms subdivide into the following classes [30]:

4.2.1 Tour Construction Heuristics

A popular and natural tour construction heuristic for the TSP is *Nearest Neighbor* algorithm. The salesman starts his journey from an arbitrary city and always travels to the nearest unvisited city. A related tour construction heuristic relies on *Greedy heuristics*. In this case, a TSP tour is viewed as a complete graph (Hamiltonian cycle) with the cities as vertices and with the distance between cities as edges. The Hamiltonian cycle is constructed by adding the available shortest edge one at a time.

There are tour construction heuristics that are called insertion heuristics. They start with a sub-tour and then extend the sub-tour by inserting the remaining cities one by one until all cities have been inserted. The starting sub-tour usually contains three cities, which form the largest triangle. The well-known insertion heuristics are nearest insertion and farthest insertion [55]. In the nearest insertion, a city among all cities not in the tour is selected such a way that its addition to the tour results the lowest increase in the length of new tour size n+1. Figure 4.1 illustrates the nearest insertion method with a 6-city TSP. In the farthest insertion, a city whose minimal distance to the tour is maximal among cities not present in the tour is added into the tour first. Although tour construction

algorithms are fast, the quality of solutions are generally not better than 10% from optimal solution [30].

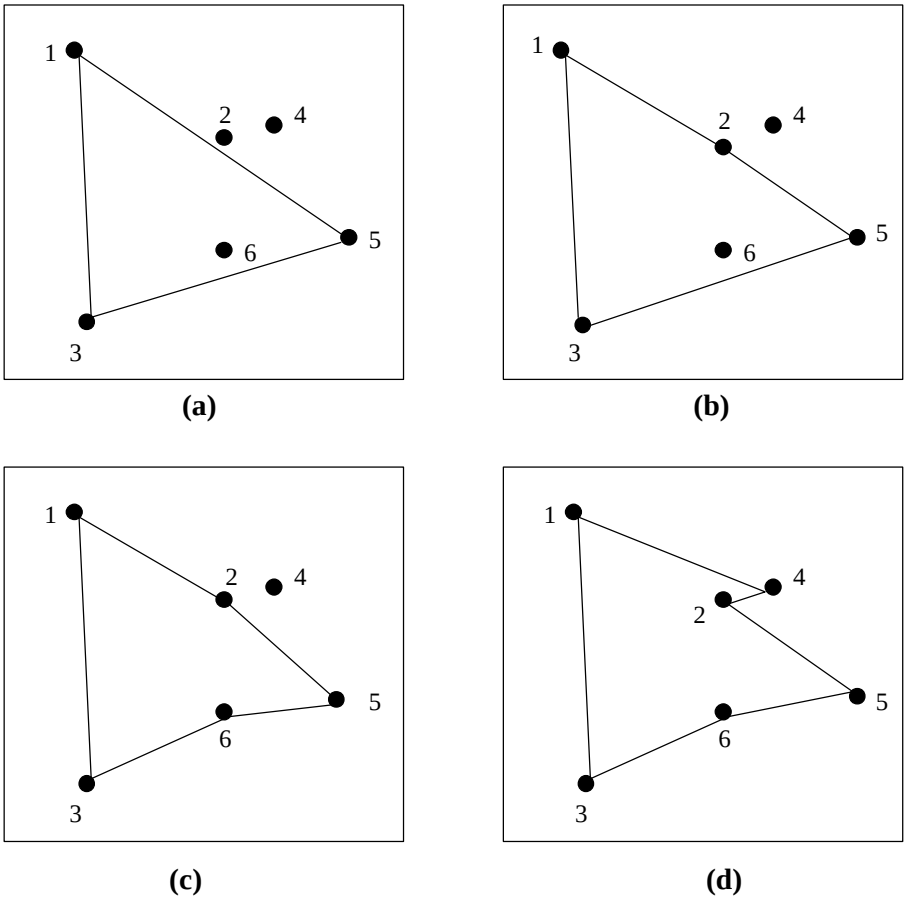


Figure 4.1 Development of a Tour on 6-city TSP using the Nearest Insertion Method

4.2.2 Local Improvement Algorithms

Local improvement algorithms for the TSP are simply based on modification of a valid tour. The well-known local improvement algorithms are 2-Opt and 3-Opt [30]. For example, a 2-Opt procedure consists of eliminating two edges and reconnecting the two resulting paths in a different way to obtain a new tour. After deleting two edges there is only one way to reconnect the paths that yield a different tour. Figure 4.2 illustrates a 2-

Opt move. The pair, which gives a shorter tour than the length of the current tour, is chosen among all pairs of edges. This procedure ends when no such pair of edges can be found. Lin & Kernighan proposed [28] an algorithm based on 2-Opt moves. This algorithm significantly reduces the search space and is known to be the most efficient local improvement algorithm [30]. Local improvement algorithms are relatively slow, but can find solution within a few percent of the optimal solution.

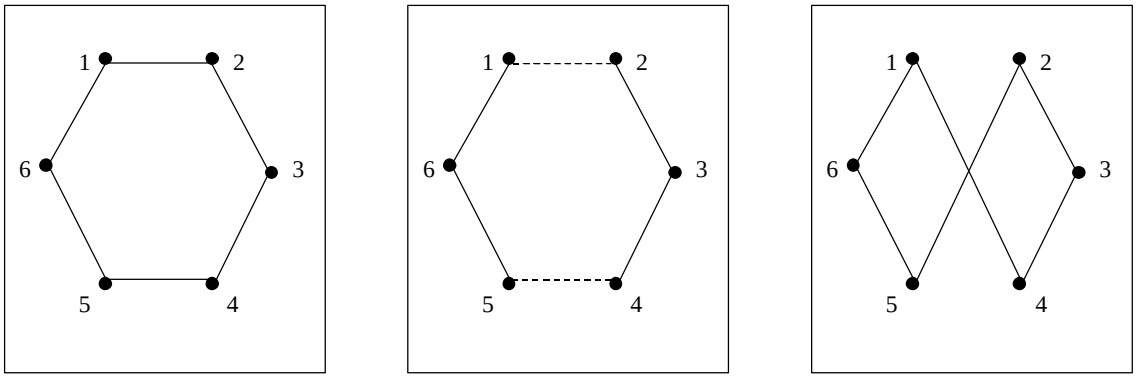


Figure 4.2 A 2-Opt Move **(a)** Original Tour **(b)** Discontinued Edges are Selected Edges for Exchange **(c)** The Resulting Tour after the 2-Opt Move

4.2.3 Memetic Algorithms

Memetic algorithms are a hybrid between genetic algorithms and a local search algorithm. A local search algorithm is applied to every individual before it is included in the population of a genetic algorithm. These algorithms can be thought of as a special kind of genetic search over the subspace of local optima [56].

Merz [57] used this approach to solve symmetric and asymmetric TSP problems and used a modified version of Lin-Kernighan algorithm [28] as local optimizer. They

reported optimal solutions for symmetric TSP instances of up to 1400 cities. These kinds of methods are called Genetic Local Search (GLS). In the GLS algorithm, after applying a genetic operator, a local search procedure is applied to the resulting individual. Therefore, all individuals in the population represent local minima.

4.2.4 TABU Search

TABU search was explained in section 2.3 and can also be used for solving the TSP. The first TABU search algorithm for the TSP was implemented by Glover [58]. His implementation starts with a tour and the tour is modified by a 2-Opt move. A TABU list is constructed by including the shorter of the two edges deleted by a 2-Opt move. Zachariasen and Dam [59] presented a new TABU search approach for the TSP problem by using 3-Opt and Lin & Kernighan [28] algorithm as a move strategy.

4.2.5 Simulated Annealing

The TSP was actually the first problem on which simulated annealing applied [13]. When simulated annealing algorithms are applied to the TSP, a valid tour sequence is modified at each step. If the length of new tour is shorter than that of current tour, the new tour will be accepted. Otherwise, the new tour will be accepted with a certain probability. Martin and Otto [60] combined simulated annealing and local search heuristics. Before applying the simulated annealing algorithm, the tour is locally optimized with a local search algorithm. Their results show that this approach is efficient for solving large TSP problems.

4.2.6 Neural Networks

Hopfield and Tank [61] proposed a Hopfield neural network to approximately solve the TSP. These neural networks consist of $\mathbf{N}^2$ neurons and $\mathbf{N}^4$ connections, where **N** is number of cities; the TSP problem is replaced by an equivalent energy minimization approach. They reported results for 10-city and 30-city TSPs, which are 5% above optimal. Kohonen's Self-Organizing Map [62] has also been applied to the TSP with modest success [31, 63]. Generally, neural networks for the TSP are slow and their effectiveness is rather modest.

4.2.7 Ant Systems

The ant algorithm models a real ant colony. Real ants have the ability to find the shortest route from a food source to the colony by exploiting pheromone information without using any visual cues [64]. When ants travel, they deposit pheromone on the ground and follow the strongest scent of pheromone previously deposited by other ants. Ant systems work in the following way. Each ant generates a complete legal tour by choosing cities based on a probabilistic rule. The rule is to visit cities, which are close to each other with a high a mount of pheromone. A certain number of artificial ants are created and placed on randomly selected cities. When each ant makes a trip to next city, the pheromone information in that edge is modified. This is called local trail updating [65]. When all the ants in the system complete their tour the ant with the shortest tour updates the edges belonging to its tour by adding an amount of pheromone trail that is inversely proportional to the tour length. This is called global trail updating [65]. There

are claims that ant system outperforms other nature-inspired algorithms such as simulated annealing and evolutionary computation [64].

4.3 Genetic Algorithms

Because finding the optimal solution for the TSP involves searching in a solution space that grows exponentially with the number of cities, solutions to the TSP have also been tried with genetic algorithms. These algorithms produce near-optimal solutions by maintaining a population of candidate solutions, which evolve by applying crossover and mutation operators under a selection scheme that biases towards the more fit individual [8]. Johnson and McGeoch [30] wrote:

> *"…in the case of the TSP many tour construction heuristics do surprisingly well in practice. The best typically get within roughly 10-15% of optimal in relatively little time. Furthermore, 'classical' local optimization techniques for the TSP yield even better results, with the simple 3-Opt heuristic getting 3-4% of optimal and the 'variable-opt' algorithms of Lin & Kernighan (1973) typically getting within 1-2%. Moreover, for geometric data the aforementioned algorithms all appear to have running time growth rates $O(N^2)$, i.e., subquadratic, at least in the range from 100 to 1,000,000 cities. These successes for traditional approaches leave less room for new approaches like tabu search, simulated annealing, etc., to make contributions. Nevertheless, at least one of the new approaches, genetic algorithms, does have something to contribute if one is willing to pay a large, although still $O(N^2)$, price in running time."*

In recent years a variety of genetic algorithms have been devised. These algorithms can be divided into two groups [66]: pure genetic algorithms and heuristic genetic algorithms.

Heuristic genetic algorithms employ genetic operators that incorporate specific information about the TSP [67-72]. Such information is generally taken advantage of operations by using heuristic algorithms such as tour construction and local improvement algorithms explained in Sections 4.2.1 and 4.2.2 in this chapter.

Pure genetic algorithms do not employ domain-specific information about the TSP and use generic genetic operators (crossover and mutation) that can be applied to arbitrary permutations [9, 73-77]. Therefore, they can be applied in any problem domain involving objects represented as permutations. These genetic operators will be explained in the following sections. In this dissertation, we will focus on pure genetic algorithms.

4.4 Representations and Genetic Operators

The key point to solve the TSP as well as any other problems using genetic algorithms is to develop an encoding that allows genetic operators to generate "*legitimate children*" without any constraint violation. GA applications to the TSP have an intrinsic problem because of the constraints imposed upon the representation for a tour: i.e., each tour must contain exactly one of instance of a city. Any omission or duplication of a city or cities leads to illegal tours. Several different tour representations and specialized genetic operators customized to these representations have been developed to solve the TSP.

4.4.1 Binary Representation

The TSP can be encoded as a binary string but it is clear that the traditional crossover and mutation operators will produce illegal tour as well. A TSP problem with ***N*** cities requires $\log_2 N$ bits to represent all cities, and a tour requires $N(\log_2 N)$ bits to be encoded. For example, for a six-city TSP a 3-bit string and a18-bit string represent cities and tours, respectively.

City Name	**A**	**B**	**C**	**D**	**E**	**F**
Binary Representation	**000**	**001**	**010**	**011**	**100**	**101**

Figure 4.3 Binary Representation of the 6-city TSP

The following two tours can be represented by binary string using Figure 4.3.

Tour 1: A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D $\rightarrow$ E $\rightarrow$ F
Tour 2: A $\rightarrow$ E $\rightarrow$ F $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C

Tour 1_{BINARY}: 000 001 010 011 100 101
Tour 2_{BINARY}: 000 100 101 001 011 010

Note that with a 3-bit binary we can represents 8 (2^3) cities, and there exist two 3-bit strings that do not correspond to any city for a 6-city TSP problem. These strings are represented by110 and 111.

Classical crossover [32] takes two tours and randomly chooses a point, where the tours are broken into two parts and then swaps the tails between the tours. For the six-city TSP tour, the classical crossover operator is illustrated in Figure 4.4.

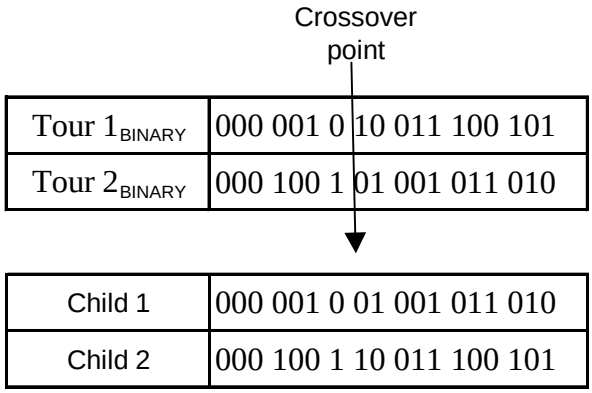


Figure 4.4 Classical Crossover Operation on the Binary Encoded 6-city TSP

According to Figure 4.4 child tour1 corresponds to (A $\rightarrow$ B $\rightarrow$ B $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C), and child tour 2 corresponds to (A $\rightarrow$ E $\rightarrow$? $\rightarrow$ D $\rightarrow$ E $\rightarrow$ F) where '?' represents non-existing city.

The classical mutation operator [32] simply flips one or more bits with a small probability. For example, the tenth bit of the Tour 1 is chosen to be mutated. Tour 1_{BINARY}: 000 001 010 **0**11 100 101. The child tour will be 000 001 010 **1**11 100 101, which corresponds to (A $\rightarrow$ B $\rightarrow$ C $\rightarrow$? $\rightarrow$ E $\rightarrow$ F) where '?' represents non-existing city.

As we have seen, classical crossover and mutation operations produce duplications and/or omissions of one or more cities. Thus, some repair algorithms are required in order to solve TSP problems with a binary encoding [40].

4.4.2 Permutation Representation

It seems natural to encode ordering problems like the TSP in permutation form [9]. Here, a tour is represented as a list of N cities. If city i is j-th element of the list, city i

is the ***j***-th city to be visited. For example, the tour B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A $\rightarrow$ E $\rightarrow$ F is simple represented by B D C A E F. In general, cities are represented as an integer number in tours.

City	A	B	C	D	E	F
Integer code	1	2	3	4	5	6

Figure 4.5 Permutation Representation for 6-city TSP

Then, the tour given above is simply represented by 2 4 3 1 5 6. This representation is also called as path representation or order representation [78]. This representation may lead to infeasible tours when traditional crossover and mutation operators are used. Many variations for crossover and mutation operators have been invented to be applicable to permutation representation.

4.4.2.1 Partially-Mapped Crossover (PMX)

PMX, proposed by Goldberg and Lingle [1], produces an offspring by choosing a portion of tour from one parent and preserving the position and relative order of as many cities as from the other parent [9]. Consider the following two tours for the illustration of the operation of PMX. Tour 1: (1 2 3 4 5 6 7 8) and Tour 2: (6 7 4 2 8 5 3 1). First, PMX selects uniformly at random two cut points along the tour. The symbol | shows the crossover points.

Tour 1: (1 2 | 3 4 5 | 6 7 8)
Tour 2: (6 7 | 4 2 8 | 5 3 1)

Second, PMX exchanges sub-strings between parents.

Offspring 1: (* * | 4 2 8 | * * *)
Offspring 2: (* * | 3 4 5 | * * *)

Third, PMX determines the mapping relationship as following:

$$3 \leftrightarrow 4 \leftrightarrow 2,\ 5 \leftrightarrow 8.$$

Then, the remaining cities are filled from original parent, if a city already present in the offspring it is replaced according to mapping relationship. The new tours will be:

Offspring 1: (1 3 | 4 2 8 | 6 7 5)
Offspring 2: (6 7 | 3 4 5 | 8 2 1)

4.4.2.2 Order Crossover (OX)

The OX, proposed by Davis [79], creates new offspring by choosing a sub-tour of one parent and preserving the relative order of cities of the other parent. Again, consider the following parent tours with two cut points market by symbol |:

Tour 1: (1 2 | 3 4 5 | 6 7 8)
Tour 2: (6 7 | 4 2 8 | 5 3 1)

The offspring are constructed in the following way. First, the tour subsequences between the cut points are inherited into the offspring, which is shown below:

Offspring 1: (* * | 3 4 5 | * * *)
Offspring 2: (* * | 4 2 8 | * * *)

Second, delete the cities, which are already present in the subsequence from the other parent.

Offspring 1 : (* * 3 4 5 * * *)	Offspring 2 : (* * 4 2 8 * * *)
Parent Tour 2: (6 7 **4** 2 8 **5** **3** 1)	Parent Tour 1 : (1 **2** 3 **4** 5 6 7 **8**)

Last, starting from the second cut point of one parent, the remaining cities are copied in the order in which they appear in the other parent. When the end of the string is reached, we continue from the first place of the string. The offspring will be:

Offspring 1: (2 8 3 4 5 1 6 7)
Offspring 2: (3 5 4 2 8 6 7 1)

The OX exploits a property of the path representation where the order of cities is important [9].

4.4.2.3 Cycle Crossover (CX)

The CX was proposed by Oliver et al. [73] and creates an offspring from the parents where every element of the offspring comes from one of the parents. This crossover satisfies that every position of the offspring must hold a value found in the corresponding position of a parent, and that the offspring are legal tours. The mechanism of CX works as follows. Consider following parent tours.

Tour 1: (1 2 3 4 5 6 7 8)
Tour 2: (6 7 4 2 8 5 3 1)

First of all, find the cycle that is defined by the corresponding positions of cities between parents starting from the first city of one of the parents. Figure 4.6 shows how to find a circle.

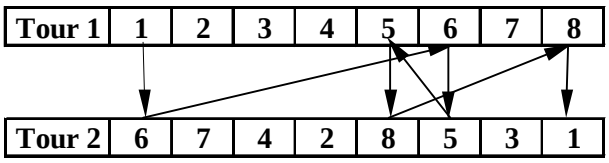


Figure 4.6 Finding a Cycle in Cycle Crossover

The circle will be $1 \rightarrow 6 \rightarrow 5 \rightarrow 8 \rightarrow 1$.

Second of all, keep the cities in the cycle corresponding positions of one parent and delete other non-cycle cities and merge the tour in order to construct an offspring.

Tour 1	**1**	*****	*****	*****	**5**	**6**	*****	**8**
Tour 2	*****	**7**	**4**	**2**	*****	*****	**3**	*****
Offspring 1	**1**	**7**	**4**	**2**	**5**	**6**	**3**	**8**

Tour 2	**6**	*****	*****	*****	**8**	**5**	*****	**1**
Tour 1	*****	**2**	**3**	**4**	*****	*****	**7**	*****
Offspring 2	**6**	**2**	**3**	**4**	**8**	**5**	**7**	**1**

The CX maintains the absolute position of the elements in the parent sequence.

4.4.3 Edge Representation

The TSP can be represented by a binary encoding based on edges on the tour [80]. Lets assume we have a six-city TSP, which can be represented by a binary encoding as a following way. Consider these two tours:

Tour 1: (A → B → C → D → E → F)
Tour 2: (B → D → C → A → E → F)

All possible edges are listed and if an edge exists in the tour; then the value is one otherwise zero. The positions of the cities in the tour are not important because the tours are circular. Also, the direction of an edge is not important in the symmetric TSP because edge AB is the same as edge BA.

	AB	**AC**	**AD**	**AE**	**AF**	**BC**	**BD**	**BE**	**BF**	**CD**	**CE**	**CF**	**DE**	**DF**	**EF**
Tour 1	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
Tour 2	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**

Figure 4.7 Binary Encoding of the TSP Based on Edges

Whitley [80] developed a genetic operator that generates good solutions for sequencing and ordering problems. This operator is called Genetic Edge Recombination. The edge recombination operator uses an edge map to create an offspring that inherits as much information as possible from the donor structures. The edge map stores all the connections from two donors that lead into and out of a city.

4.4.4 Adjacency Representation

A tour is represented as a list of N cities in adjacency representation. The city j is in the position i in the list if and only if the tour leads from city i to city j. For example, the tour $(1 \rightarrow 3 \rightarrow 7 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 6 \rightarrow 8)$ is represented by (3 5 7 6 4 8 2 1). Although each tour has unique adjacency representation, some of them may represent illegal tours. For example, the adjacency list (3 5 7 6 2 4 1 8) leads to the premature partial tour $(1 \rightarrow 3 \rightarrow 7 \rightarrow 1)$. It is clear that the adjacency representation does not support the classical crossover operation and a repair algorithm might be necessary in order to correct illegal tours. Some crossover operators were proposed for adjacency representation. These are alternating-edge crossover, subtour-chunks crossover, and heuristic crossover [8].

The main advantage of the adjacency representation is that it allows schemata analysis. However, this representation generally produces poor results for all crossover operators listed above [8].

4.4.5 Ordinal Representation

In ordinal representation, a tour with ***N***-cities is represented as a list of ***N*** elements, where the ***i***-th element of the list is a number in the range from 1 to ***N***-***i***+1. Here, there exists an ordered list of cities ***R***, which serves as a reference point. For example, assume ***R*** = (1 2 3 4 5 6 7 8) and the tour ***T*** = (1 $\rightarrow$ 3 $\rightarrow$ 7 $\rightarrow$ 2 $\rightarrow$ 5 $\rightarrow$ 4 $\rightarrow$ 6 $\rightarrow$ 8) is represented by ***O*** = (1 2 5 1 2 1 1 1). This order representation ***O*** can be transformed to original tour. The first element in ***O*** is 1, which means that takes the first element of ***R***, which is city 1 and removes it from ***R***. The new is ***R*** = (2 3 4 5 6 7 8). The first element of the tour **T** will be city 1. Look at the second element of ***O***, which is 2. This means that we take the second element of R, which is 3 and remove from R. The second city in the tour is city 3. If we continue this way until all elements of R are removed, we will get the original route.

The main advantage of ordinal representation is that the classical one point crossover does not need a repair algorithm. Although partial tours to the left of the crossover point do not change, the partial tour to the right of the crossover point is changing in a quite random way. That is the reason why this representation produces poor results with classical one point crossover operator [8].

4.4.6 Random Keys Representation

Bean [77] proposed the Random Key representation. This representation uses random numbers between 0 and 1 to represent a solution. These numbers are used as sort keys to decode a solution. For example, consider a 6-city tour represented by a random key where random key tour is represented as (0.21 0.85 0.43 0.71 0.91 0.10). The random

number at position ***i*** determines the visiting order when we sort them in ascending order. This random key tour represents the following tour ($6 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5$). The important advantage of random key is that all offspring resulting from by crossover are feasible solutions.

4.4.7 Matrix Representation

Fox and McMahon [81] proposed a matrix representation for the TSP. In this matrix, the element in row ***i*** and column ***j*** is 1 if and only if the city ***i*** occurs before city ***j***. For example, the tour ($2 \rightarrow 3 \rightarrow 1 \rightarrow 4$) is represented by the following matrix.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

In this representation, the matrix representing a tour has the following properties [8]:

- Total number 1s = $\frac{n(n-1)}{2}$
- The diagonal elements of the matrix are all zero. $\boldsymbol{m_{ij}} = 0$ for $0 \leq \boldsymbol{i} \leq \boldsymbol{N}$.
- If $\boldsymbol{m_{ij}} = 1$ and $\boldsymbol{m_{jk}} = 1$ then $\boldsymbol{m_{ik}} = 1$.

If the first conditions does not hold and the other two hold, then cities are partially ordered and we can complete such a matrix to get a legal tour.

4.5 Modified Partially Mapped Crossover

Partially Mapped Crossover (PMX) [1] was explained in section 4.4.2.1 in this chapter. We will first illustrate and explain the classic PMX crossover operator and then,

we will propose a modification into Partially Mapped Crossover in order to increase the efficiency of the crossover operator by reducing the premature convergence.

4.5.1 PMX and Premature Convergence

PMX produces an offspring by choosing a portion of tour from one parent and preserving the position and relative order of as many cities as from the other parent [9]. Consider the following two tours for the illustration of the operation of PMX.

Tour 1: (1 2 3 4 5 6 7 8)
Tour 2: (6 7 4 2 8 5 3 1)

First, PMX selects uniformly at random two cut points along the tour. The symbol | shows the crossover points.

Tour 1: (1 2 | 3 4 5 | 6 7 8)
Tour 2: (6 7 | 4 2 8 | 5 3 1)

Second, PMX exchanges sub-strings between parents.

Offspring 1: (* * | 4 2 8 | * * *)
Offspring 2: (* * | 3 4 5 | * * *)

Third, PMX determines the mapping relationship as following:

$$3 \leftrightarrow 4 \leftrightarrow 2,\ 5 \leftrightarrow 8.$$

Then, the remaining cities are filled from original parent, if a city already exists in the offspring it is replaced according to mapping relationship. The new tours will be:

Offspring 1: (1 3 | 4 2 8 | 6 7 5)
Offspring 2: (6 7 | 3 4 5 | 8 2 1)

Goldberg and Lingle [1] reported results for 10-city and 33-city TSP problems. They used roulette wheel selection (fitness proportional selection), inversion operation for mutation, PMX with crossover probability 0.60, population size 200 and reported

optimal solution for 10-city TSP. For 33-city TSP, using population size 2000, they reported solutions within 10 percent of the optimal solution.

The principal natural mechanism responsible for recoding a problem is the inversion operator [9]. The inversion operation involves selecting two points within a chromosome and reversing the substring between these points. This operation produces legal offspring and useful finding good string ordering in TSP like problems [32].

Frantz [82] used two ways of choosing inversion points: linear and linear+end. The linear inversion method, which is same as classical inversion operation, chooses two points at random (i.e., each point has an equal likely probability of being chosen) and all genes between and including these points are reverted. Frantz calculated the probability of any position ***m*** being inverted on a string length ***N***.

$$\text{Probability\{gene moved\}} = \frac{2[m(N+1) - m^2 - 1]}{N^2}$$

Frantz [82] reported that when ***N*** is 25, genes located in central positions are almost seven times more likely to be included in an inversion operation than the genes located in either of the two ends. This situation makes it difficult for any gene located in one of the end position to be close to a gene located on the other end. He devised the linear+end inversion method in order to alleviate this problem. In linear+end inversion, cut points are chosen in the same way as in linear inversion but the inversion is performed either between cut points or between first cut point and the left end of the string or between second cut point and the right of the string with a certain probability.

Another way for reducing these end effects is to treat the chromosome as a ring, with no beginning and no end [9]. In this case, each location is equally likely to be relocated under a single inversion.

Premature convergence is one of the major handicaps for genetic algorithms. It has been observed that this problem is closely related to the problem of losing diversity in the population. Individuals with high fitness (super individuals) take over the population after some number of generations. One of the reasons for premature convergence is related to the end effect [9, 82].

4.5.2 A Modification to PMX

Our experiments have been shown that PMX operator causes the premature convergence of genetic algorithms for the TSP because of the end effect. First, PMX is implemented same as in Goldberg and Lingle [1]. In order to reduce the end effect, we treated one of the parent chromosomes as a ring, with no beginning and no end before crossover occurs. In this implementation of the genetic algorithm, we used the path representation and started with a random population. However, there is no priori reason to believe that a non-random population will not be appropriate [83].

The individuals are selected for next generation based on roulette wheel selection proportionally to their rank rather than actual evaluation values [8]. The rank probability for selecting an individual for the next generation is calculated based on following formula.

$$p(i) = q(1-q)^{i-1} \ ,$$

where ***p(i)*** is the rank probability of tour ***i***. ***i* = 1…*pop_size***. ***i*** = 1 represents the best individual, while ***i* = *pop_size*** represents the worst tour.

The classical inversion operation is employed for mutation. In this algorithm for the TSP all individuals are valid tours. For example, we can represent 5-city TSP as following.

1	2	3	4	5

Because a tour is a closed loop, the following tours are also exactly same as above.

2	3	4	5	1
3	4	5	1	2
4	5	1	2	3
5	1	2	3	4

We will call this virtual representation a dynamic representation. If we cross over two of same individual without dynamic representation, we get a twin pair offspring, where the children are exactly same as parents. Let's assume crossover points 1 and 3.

Parent 1	1	2	3	4	5
Parent 2	1	2	3	4	5

Child 1	1	2	3	4	5
Child 2	1	2	3	4	5

If we allow first parent becomes one of its states with probability $\frac{1}{N} = \frac{1}{5} = 0.20$, where N is the number of city. Lets assume that first parent found in the following state.

Parent 1	**2**	**3**	**4**	**5**	**1**
Parent 2	**1**	**2**	**3**	**4**	**5**

If we apply crossover operation onto the parents, assume crossover points 1 and 3, we get:

Parent 1	**2**	**3**	**4**	**5**	**1**
Parent 2	**1**	**2**	**3**	**4**	**5**

Parent 1	**2**	**2**	**3**	**5**	**1**
Parent 2	**1**	**3**	**4**	**4**	**5**

$2 \leftrightarrow 3 \leftrightarrow 4$

Child 1	**4**	**2**	**3**	**5**	**1**
Child 2	**1**	**3**	**4**	**2**	**5**

As we have seen above, even though the parent donors are twins, they can produce different children.

4.5.3 Comparison of Classic PMX and Modified PMX

We have studied four Euclidean TSPs. Three of them, berlin52, eil76, and rd100, are from the TSPLIB [84]. The last problem, oliver30, is from Oliver et al. [73]. Each of the problems is solved 10 times by two genetic algorithms. The only difference between two algorithms is the representation of the first parent before PMX. The genetic algorithm with the modified PMX is called as Dynamic PMX and the algorithm with classical PMX is called as Static PMX. These results are presented in the Tables 4.1, 4.2, 4.3, and 4.4 for 30, 52, 76, 100-city TSPs, respectively.

Table 4.1 Results of Static and Dynamic PMX for 30-city Problem

Oliver30								
Number of Cities	: 30		Population Size	: 150		Selection Pressure	: 0.08	
Crossover Probability	: 0.9		Mutation Probabiliy	: 0.02		The Best Known Solution	: 423.741	
Run	STATIC PMX				DYNAMIC PMX			
	Tour Length	Run Time	Generation	Difference	Tour Length	Run Time	Generation	Difference
1	457.816	18	1276	-8.04%	423.741	24	1612	0.00%
2	424.692	24	1738	-0.22%	423.912	24	1666	-0.04%
3	425.104	18	1334	-0.32%	423.741	22	1550	0.00%
4	458.496	24	1709	-8.20%	423.741	29	2065	0.00%
5	428.673	20	1393	-1.16%	424.692	18	1249	-0.22%
6	462.812	19	1335	-9.22%	438.382	21	1472	-3.46%
7	494.499	20	1467	-16.70%	423.741	20	1386	0.00%
8	455.520	23	1667	-7.50%	438.382	36	2478	-3.46%
9	439.786	24	1727	-3.79%	424.692	18	1230	-0.22%
10	461.058	22	1626	-8.81%	423.741	18	1250	0.00%
Average	**450.846**	**21**	**1527**	**-6.40%**	**426.877**	**23**	**1596**	**-0.74%**

Table 4.2 Results of Static and Dynamic PMX for 52-city Problem

Berlin52								
Number of Cities	: 30		Population Size	: 250		Selection Pressure	: 0.08	
Crossover Probability	: 0.9		Mutation Probabiliy	: 0.02		The Best Known Solution	:7544.37	
Run	STATIC PMX				DYNAMIC PMX			
	Tour Length	Run Time	Generation	Difference	Tour Length	Run Time	Generation	Difference
1	8175.130	199	2262	-8.36%	7544.370	192	2195	0.00%
2	8110.700	154	1764	-7.51%	7863.300	163	1863	-4.23%
3	7966.550	202	2262	-5.60%	7544.370	221	2470	0.00%
4	8145.900	191	2160	-7.97%	7816.430	218	2495	-3.61%
5	8255.190	216	2480	-9.42%	7777.330	155	1796	-3.09%
6	7971.170	277	3183	-5.66%	7544.370	181	1958	0.00%
7	8288.520	158	1813	-9.86%	7887.230	371	4205	-4.54%
8	8287.730	298	3410	-9.85%	8040.450	153	1739	-6.58%
9	8173.780	198	2279	-8.34%	7973.310	168	1933	-5.69%
10	8425.190	200	2309	-11.68%	7777.330	252	2894	-3.09%
Average	**8179.986**	**209**	**2392**	**-8.43%**	**7776.849**	**207**	**2355**	**-3.08%**

Table 4.3 Results of Static and Dynamic PMX for 76-city Problem

Eil76								
Number of Cities	: 76		Population Size	: 300		Selection Pressure	: 0.08	
Crossover Probability	: 0.9		Mutation Probabiliy	: 0.02		The Best Known Solution	: 545.39	
Run	STATIC PMX				DYNAMIC PMX			
	Tour Length	Run Time	Generation	Difference	Tour Length	Run Time	Generation	Difference
1	598.012	708	2465	-9.65%	550.279	1126	3922	-0.90%
2	610.988	1208	4197	-12.03%	564.321	785	2741	-3.47%
3	590.146	657	2284	-8.21%	551.941	953	3333	-1.20%
4	597.797	1089	3788	-9.61%	558.566	1745	6077	-2.42%
5	594.241	1016	3543	-8.96%	555.320	825	2874	-1.82%
6	594.520	990	3438	-9.01%	563.637	787	2744	-3.35%
7	618.222	777	2710	-13.35%	560.371	1091	3824	-2.75%
8	611.990	1073	3735	-12.21%	562.285	972	3388	-3.10%
9	595.472	847	2956	-9.18%	558.251	1148	4023	-2.36%
10	608.441	808	2521	-11.56%	560.739	997	3430	-2.81%
Average	**601.983**	**917**	**3164**	**-10.38%**	**558.571**	**1043**	**3636**	**-2.42%**

Table 4.4 Results of Static and Dynamic PMX for 100-city Problem

Rd100								
Number of Cities	: 100	Population Size	: 350	Selection Pressure	: 0.08			
Crossover Probability	: 0.9	Mutation Probabiliy	: 0.02	The Best Known Solution	: 7910.40			
Run	STATIC PMX				DYNAMIC PMX			
	Tour Length	Run Time	Generation	Difference	Tour Length	Run Time	Generation	Difference
1	8945.91	4841	6347	-13.09%	8290.42	3361	4600	-4.80%
2	8610.15	3358	4592	-8.85%	8404.97	3338	4574	-6.25%
3	8371.87	2639	3459	-5.83%	8115.11	4345	5971	-2.59%
4	8321.18	3386	4625	-5.19%	8432.02	3342	4543	-6.59%
5	9080.16	3694	5065	-14.79%	8286.90	4073	5527	-4.76%
6	9147.68	2774	3801	-15.64%	8126.83	5672	7757	-2.74%
7	8943.40	2860	3886	-13.06%	8298.87	5328	7284	-4.91%
8	8586.54	2980	4072	-8.55%	8257.43	5057	6875	-4.39%
9	8677.63	4008	5494	-9.70%	8245.67	3769	5180	-4.24%
10	8557.58	3309	4538	-8.18%	7942.80	4171	6429	-0.41%
Average	**8724.21**	**3385**	**4588**	**-10.29%**	**8240.10**	**4246**	**5874**	**-4.17%**

Table 4.5 presents overall comparison results of static and dynamic PMX for the four TSPs. According to computational results, the proposed method improves solution quality very well. The results are encouraging because this algorithm is a pure genetic algorithm, which does not use knowledge-augmented operators.

Table 4.5 Summary Results for Static and Dynamic PMX

Data Set	Number of Cities	Population Size	Crossover Probability	Mutation Probability	Selection Pressure	% Difference from Optimal	
						Static PMX	Dynamic PMX
oliver30	30	150	0.9	0.02	0.08	-6.40	-0.74
berlin52	52	250	0.9	0.02	0.08	-8.43	-3.08
eil76	76	300	0.9	0.02	0.08	-10.38	-2.42
rd100	100	350	0.9	0.02	0.08	-10.29	-4.17

4.6 Evolutionary Programming Approach to the TSP

Evolutionary Programming (EP) has been introduced in the section 2.4.2 in Chapter 2. First, we will revisit evolutionary programming and explain it in a detail. Second, a new algorithm for solving the TSP is developed based on EP in which a rank based and simulated annealing selection scheme are implemented.

4.6.1 Evolutionary Programming

Evolutionary programming (EP) originated from Lawrence Fogel in the late 1960s [85] and is a stochastic optimization method similar to genetic algorithms. It originally aimed at evolution of artificial intelligence in the sense of developing ability to predict changes in environment [22, 23]. The key difference between GAs and EPs is that there is no more crossover operator and that selection is based on a survival of the fittest criterion. In an application of EA, after initializing the population, all $\boldsymbol{N}$ individuals are selected to be parents. Only mutation is used for producing $\boldsymbol{N}$ children from $\boldsymbol{N}$ parents and $\boldsymbol{N}$ survivors are chosen from $2\boldsymbol{N}$ individuals (parents plus children), using a probabilistic function based on fitness. In other words, individuals with a greater fitness have a higher chance to present in the next generation. Figure 4.8 shows the outline of the EP.

EP was used in [86] to solve three test traveling salesman problems (30-city, 50-city and 75-city in [40]). They reported better results than the previously known best solutions. Each tour was encoded as a list of cities to be visited in order. The mutation operator selected two positions along the tour and reversed the order of cities between these positions. The population consisted of 100 parent tours and each parent produced a single offspring through mutation operator. 100 tours were selected to be parents for next generation based on a competition rule [87]. Competition for survival proceeds as following. Each tour competes against 10% of the population. The probability of winning in each encounter is equal to the opponent's tour length divided by the sum of the two competing tour lengths. Also, the population size is forced to decrease linearly over time

(one tour per 5000 evaluated offspring) in order to simulate a decrease in natural resources in the environment [87].

```
Procedure evolutionary programming
Begin
        Choose a coding to represent variables
        t ← 0
        Initialize population P(t)
        Evaluate population P(t)

        While (not termination condition) do
                t ← t+1
                Select P(t)
                Alter P(t) with mutation and produce C(t)
                Evaluate P(t) and C(t)
                Survive P(t) from P(t) + C(t)
        End
End
```

Figure 4.8 Standard Evolutionary Programming Algorithm

4.6.2 Evolutionary Programming with Constant Population (EPC)

We propose a new algorithm based on EP. The proposed algorithm, Evolutionary Programming with Constant Population (EPC) is different than classic or previously used EP. First, EPC always keeps a constant number of individuals similar to genetic algorithms. Second, EPC uses a selection scheme (simulated annealing) with mutation operation. The procedure for EPC is shown in Figure 4.9. Details are for procedure of EPC given below.

```
Procedure evolutionary programming based on simulated annealing
Begin
        Choose a coding to represent variables
        t ← 0
        Initialize population P(t)
        Evaluate population P(t)

        While (not termination condition) do
                t ← t+1
                Select P(t)
                Mutate P(t) with some probability.
                Evaluate P(t)
        End
End
```

Figure 4.9 Evolutionary Programming with Constant Population

Representation and Initialization: The path representation was employed for the representation of the tour in the population. As an initial population a number of random permutations (tours) equal to population size is created. The population size is always constant. This means that if an individual is created it immediately replaces with its parent or dies off. In classic EP, each individual in the population creates an offspring. Both parents and offspring will survive until a selection (survival of the fittest) operation is performed, which reduce the size of the population half.

Evaluation and selection: The individuals are evaluated and ranked from the shortest route to the longest one. Selection of the individuals for the next generation is based on roulette wheel selection with proportionality to their rank as explained in previous section [8].

Mutation: Following the idea of EP (in the narrow sense) the population is altered by mutation only (i.e., no crossover operation presents). Therefore, this algorithm

can be interpreted as the asexual counterpart the genetic algorithms. The tours are mutated with a simple inversion operator. The simple inversion mutation randomly determines two cities in a tour and reverses the cities between them [32]. After creating offspring a tournament takes place between the parent tour and its offspring. The selection process uses simulated annealing based on a Boltzman probability distribution. The offspring can win the tournament based on Boltzman probability distribution.

$$prob(select) = \exp\left(\frac{p_i - c_i}{k_B T}\right),$$

where,

$\boldsymbol{p_i}$ and $\boldsymbol{c_i}$ are the length of the parent tour and the length of the child tour, respectively.

$\boldsymbol{k_B}$ and $\boldsymbol{T}$ represent the Boltzman constant and temperature, respectively.

Notice that if the length of offspring tour is shorter than that of parent ($\boldsymbol{p_i} \geq \boldsymbol{c_i}$); then, the child always will be replaced with its parent. If the length of offspring is longer than that of its parent ($\boldsymbol{p_i} \leq \boldsymbol{c_i}$); then, the child will be replaced with its parent with some probability. This acceptance rule is known as the Metropolis criterion [13].

4.6.3 Experimental Results for EPC

EPC algorithm was applied to four standard TSPs: 30-city from [73], 50- and 75-city from [88] and 100-city (KroA100) from TSPLIB. Each of the problems solved 10 times by EPC with different random population initializations. The results and parameters used for these problems are presented in Tables 4.6, 4.7, 4.8, and 4.9. We used two different calculations for the tour lengths: the real and the integer tour distance. The difference between the real and integer tour lengths is that in the first case distances

between cities are measured by floating point approximations of real numbers. On the other hand, in the integer case the distance between cities are calculated as an integer in following way:

$\mathbf{X}= \mathbf{X_i}-\mathbf{X_j}$, where **X** is x coordinate of the city.

$\mathbf{Y}= \mathbf{Y_i}-\mathbf{Y_j}$ where **Y** is y coordinate of the city.

$\mathbf{D_{ij}}=\text{int}(\text{sqrt}(\mathbf{X}^2+\mathbf{Y}^2)+0.5)$, where $\mathbf{D_{ij}}$ is the distance between city **i** and city **j**.

Table 4.6 Results of EPC for 30-city TSP

Oliver30						
The Best Known Solution: 423.741 (real) and 420 (integer)						
Number of Cities : 30			Selection Pressure : 0.06			
Population Size : 450						
Run	**Tour Length**				**Difference**	
	Real	**Integer**	**Run Time**	**Generation**	**Real**	**Integer**
1	423.741	420	10	262	0.00%	0.00%
2	424.635	421	7	187	-0.21%	-0.24%
3	423.912	420	8	203	-0.04%	0.00%
4	423.741	420	7	196	0.00%	0.00%
5	424.573	422	6	156	-0.20%	-0.48%
6	423.741	420	8	205	0.00%	0.00%
7	423.741	420	8	198	0.00%	0.00%
8	423.741	420	14	366	0.00%	0.00%
9	423.741	420	7	176	0.00%	0.00%
10	423.741	420	8	214	0.00%	0.00%
Average	423.931	420	8	216	-0.04%	-0.07%

Table 4.7 Results of EPC for 50-city TSP

Eil50						
The Best Known Optimal Solution: 427.855 (real) and 425 (integer)						
Number of Cities : 50			Selection Pressure : 0.08			
Population Size : 1000						
Run	**Tour Length**				**Difference**	
	Real	**Integer**	**Run Time**	**Generation**	**Real**	**Integer**
1	427.312	426	33	218	0.13%	-0.24%
2	436.970	436	25	173	-2.13%	-2.59%
3	441.301	439	43	298	-3.14%	-3.29%
4	428.856	428	50	344	-0.23%	-0.71%
5	427.312	426	37	257	0.13%	-0.24%
6	433.342	432	37	259	-1.28%	-1.65%
7	427.312	426	64	431	0.13%	-0.24%
8	428.178	426	27	182	-0.08%	-0.24%
9	427.779	427	34	229	0.02%	-0.47%
10	433.370	430	33	227	-1.29%	-1.18%
Average	431.173	430	38	262	-0.78%	-1.08%

Table 4.8 Results of EPC for 75-city TSP

Eil75						
The Best Known Optimal Solution: 542.37 (real) and 535 (integer)						
Number of Cities : 75			Selection Pressure : 0.08			
Population Size : 1250						
Run	Tour Length				Difference	
	Real	Integer	Run Time	Generation	Real	Integer
1	548.678	543	83	344	-1.16%	-1.50%
2	553.768	548	101	417	-2.10%	-2.43%
3	560.196	553	83	343	-3.29%	-3.36%
4	557.932	552	107	441	-2.87%	-3.18%
5	550.924	546	187	734	-1.58%	-2.06%
6	561.051	558	65	267	-3.44%	-4.30%
7	561.758	556	61	253	-3.57%	-3.93%
8	560.247	553	70	287	-3.30%	-3.36%
9	553.297	547	98	410	-2.01%	-2.24%
10	553.970	552	84	345	-2.14%	-3.18%
Average	556.182	551	94	384	-2.55%	-2.95%

Table 4.9 Results of EPC for 100-city TSP

KroA100						
The Best Known Optimal Solution: 21285.44 (real) and 21282 (integer)						
Number of Cities : 100			Selection Pressure : 0.07			
Population Size : 1500						
Run	Tour Length				Difference	
	Real	Integer	Run Time	Generation	Real	Integer
1	21967.100	21967	273	622	-3.20%	-3.22%
2	21921.800	21919	158	351	-2.99%	-2.99%
3	22030.400	22029	184	434	-3.50%	-3.51%
4	21900.200	21898	165	397	-2.89%	-2.89%
5	21906.500	21905	196	492	-2.92%	-2.93%
6	21981.900	21979	258	640	-3.27%	-3.28%
7	21787.200	21786	174	426	-2.36%	-2.37%
8	21930.400	21927	239	595	-3.03%	-3.03%
9	22213.100	22212	259	600	-4.36%	-4.37%
10	21828.900	21829	170	420	-2.55%	-2.57%
Average	21946.750	21945	208	498	-3.11%	-3.12%

The comparison of EPC with other heuristics (Genetic Algorithm, Evolutionary Programming, and Simulated Annealing) is performed on the best results because most of the available studies do not the give average results. The comparisons are performed on both integer and real tour length for each of the TSPs. Table 4.10 gives the references of the compared methods.

Table 4.10 References of Compared Methods

Problem Name	Genetic Algorithm	Evolutionary Programming	Simulated Annealing
Oliver30	(Oliver, 1987)	(Fogel, 1993)	N/A
Eil50	(Whitley, 1989)	(Fogel, 1993)	(Lin, 1993)
Eil75	(Whitley, 1989)	(Fogel, 1993)	(Lin, 1993)
KroA100	(Whitley, 1989)]	N/A	N/A

Table 4.11 Comparison of EPC with Other Methods

Problem Name	Solution Type	Proposed EPC	Genetic Algorithm	Evolutionary Programming	Simulated Annealing	The Best Known Solution
Oliver30	integer	420	N/A	N/A	N/A	N/A
	real	**423.741**	**423.741**	**423.741**	N/A	423.741
Eil50	integer	**426**	428	**426**	443	425
	real	**427.312**	N/A	427.855	N/A	N/A
Eil75	integer	543	545	**542**	580	535
	real	**548.678**	N/A	549.81	N/A	N/A
KroA100	integer	21786	**21761**	N/A	N/A	21282
	real	21787.2	N/A	N/A	N/A	N/A

According to the TSP benchmark problems, the use of EPC brings significant improvement. The 30-city problem is solved the most easily. The proposed method has found optimal solution in seven of ten runs. When the number of cities increases the performance of the method decreases. The comparison of results shows that the proposed method performs very well for 30, 50 and 75 cities, which gets shorter routes in terms of real-valued tour distances. For 100-city problem, the result is worse than the result of GA but the proposed method found 3.12% worse than the optimal solution in 10 runs.

CHAPTER 5

Evolutionary Algorithms for Predictive Modeling and Data Mining

5.1 Predictive Modeling and Data Mining

Advances in data collection and storage technologies have provided large amounts of data for business, government, and scientific purposes. [89]. Due to the large amount of data that are stored in databases, traditional data analysis tools are often not well suited to extract knowledge from these databases. Consequently, new techniques and tools have been devised with the aim to automatically transform data into useful information and knowledge. This emerging research area is called as Data Mining. Data mining is a field of study that deals with extracting knowledge and useful information from large databases, without putting restrictions on the amount or types of data in a database [90]. The goal of the data mining can be broadly divided into two categories: prediction and description [91]. A prediction task involves using attributes of a database to be able to predict on unknown future values of a dependent variable of interest. On the other hand, the description task focuses on interpreting the data. Figure 5.1 illustrates a decision-making process for data mining [91].

5.1.1 Standard Data Mining Problems

In this dissertation, a predictive data mining problem is defined as the *standard data mining* problem. In this context, a standard data mining problem is a multivariate

regression or classification problem for which there are many candidate features to choose from in order to generate a predictive model. The ultimate aim is not just to build a good predictive model but also explain and interpret to some degree how and why the model works. The standard data mining problem can be different from a standard statistical regression approach in the sense that the number of descriptive features for a predictive data mining problem can be extremely large even for data sets with a relatively small number of data records. [92]. The standard data mining problem is different from feature selection in statistics in the sense that a typical data mining problem might deal with multiple sources of large datasets (i.e., large either in number of patterns, or in number of features, or both) with potentially missing, false and conflicting data. The data mining approach is also different from the statistical approach in the sense that the data sets for data mining problem are often too large to fit in memory or have so many features that traditional statistical methods might not apply (i.e., curse-of-dimensionality problem).

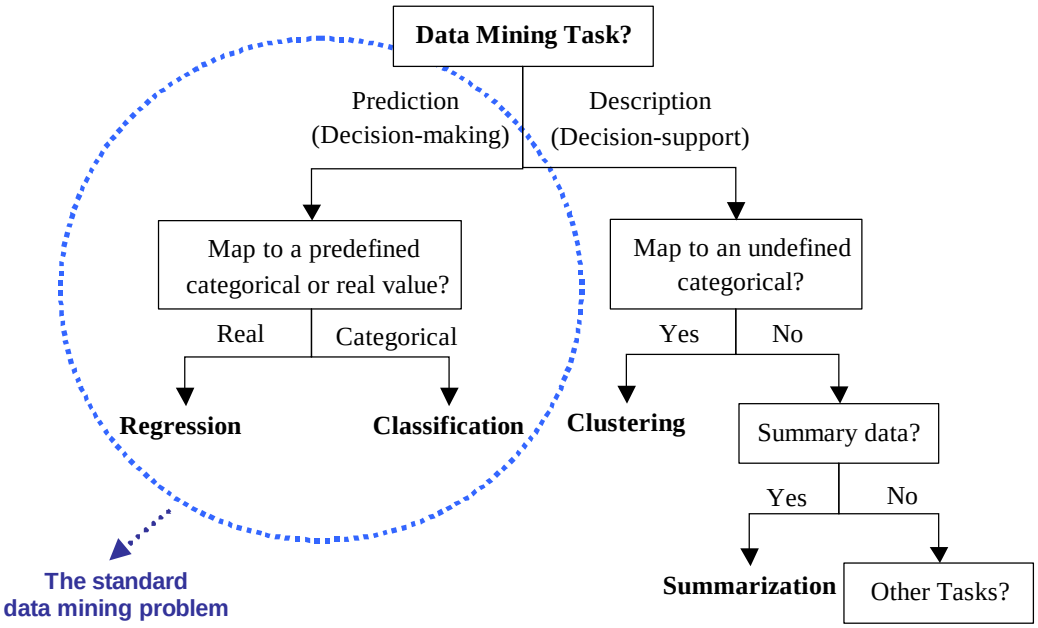


Figure 5.1 The Standard Data Mining Problem

5.1.2 Data Strip Mining Problems

As the number of features becomes larger than the number of data points, the standard data-mining task becomes more challenging and complex. Most predictive modeling approaches do not work well in these situations. One solution is to collect more data, but this option is not always available. In this case, a data analyst must extract as much information as possible from the existing data. In this context, the standard data mining problem is defined as the process of building good predictive model based on a relevant subset of the descriptive features that can help explain the model. The extreme case, where the number of features is on the order of or greater than the number of data points, is defined here as a data strip mining problem [92]. In this case, the challenge is to find a subset of features that provides a good predictive model.

Feature selection is a common task in many classification and regression problems. Feature selection involves minimizing the number of relevant features and maximizing the predictive power of the model. From this point of view feature selection can be viewed as a special type of multi-objective optimization problem [93].

Predictive data mining has an objective to predict a dependent variable **Y** (also called the output or the response variable) based on a number of features represented by $\mathbf{X} = (x_1, \ldots, x_n)$ (also called the independent, input or predictor variables). In machine learning the classical supervised learning task involves the use of learning algorithms for training on data where both the features (**x**) and corresponding dependent variable (**y**) are known. The goal of such a learning algorithm is often to use this data set in order to derive a predictive model for ***y*** that can ultimately be explained by a set of rules that involves a subset of features.

The strategies of learning algorithms depend upon the nature of the response variable in terms of the types of values it can be assigned. The most common data types are categorical and ordinal. In ordinal response variable cases, values for the response variable are real numbers and there is an order relationship possible between every pair of responses that can be characterized by a distance measure. These kinds of predictive modeling problems are called ***regression*** problems. In the categorical response cases, the response variable ***y*** realizes an unordered discrete value only and there is no defined order relationship or distance measure between a pair of response values. Two response variables are either equal (same) or not equal (different class). These kinds of problems are called ***classification*** problems [94].

In principle, all input features (including irrelevant features) are used to approximate the underlying function between the response variable and the features. In practice, the presence of irrelevant features can cause several problems [92, 95]:

1. The irrelevant input features will increase the dimension of the problem and will require greater computational cost.
2. Irrelevant features lead to a curse-of-dimensionality problem.
3. The irrelevant input features may lead to overfitting.
4. Excess features make the model more difficult to predict.
5. Excess features make the model more difficult to explain.

The key motivation for feature selection is to choose a subset of $\mathbf{X_S}$ of the complete set of input features $\mathbf{X}=\{x_1, x_2, \ldots, x_N\}$ so that this subset $\mathbf{X_S}$ can estimate the output $\mathbf{Y}$ with an accuracy higher than or comparable to the performance of the complete input set $\mathbf{X}$ (preferably, with a reduced computation time)[95].

This dissertation, mainly concerns **feature selection** for **regression problems**, but the methodology should be generally applicable. Feature selection has been addressed to some degree in statistics, pattern recognition, econometrics, computational chemistry, and machine learning, but still remains an interesting and difficult problem (depending on the particular application). The machine learning community has recently targeted it and has developed its own techniques [6]. Feature selection still remains as a difficult problem for all of the study domains.

5.2 In-Silico Drug Design

An important area of the computational chemistry is the construction of a predictive relationship between features related to chemical structure and certain activity response variables (i.e. a measure of how organic chemical compounds interact with and affect certain living organisms). The design of a drug with desired pharmaceutical properties is an important and challenging task that involves evaluating and searching a large number of potential candidate molecules with regard to their pharmaceutical properties [96]. Traditionally, the introduction of a novel drug was done by trial-and-error, which involves synthesizing and testing a large numbers of diverse compounds. This is time consuming, costly, and laborious. The design of a new drug before it hits the market typically requires 10 to 15 years of research and development (R&D) and requires on the order of $500,000 of resources per drug [97].

Recent developments in high-throughput chemistry enabled the synthesis of a large number of molecular compounds (specifically several thousands of molecules within a few days) [98]. The idea behind rational drug design is to utilize large existing

pharmaceutical databases to derive structure/activity correlation models that can exploit and identify novel relationships between the molecular structure and the pharmaceutical properties. Each molecule in a dataset is characterized by an appropriate set of descriptive features or descriptors. These descriptors will then be used to build Quantitative Structure-Activity Relationship (QSAR) models that characterize and predict relevant biological responses or pharmaceutical properties. These QSAR models will then be used to screen existing molecular databases for novel drug candidates or to create new virtual combinatorial libraries of potentially active compounds [97].

The basic idea behind QSAR, first introduced by Hansch et al. [2], is to predict the biological activity of new untested chemicals from the knowledge of their chemical structures. QSAR assumes that the change in biological activity that is observed within a series of similar compounds is a function of the change in chemical structure within the series [99]. Thus, QSAR methods deal with identifying important structural features of molecules that are relevant to explain variations in biological or chemical properties. Most of the QSAR methods developed since Hansch et al. [2] dealt with descriptors of molecular structures derived from a two-dimensional (2D) representation of molecular structures (i.e., based on molecular connectivity).

The rapid accumulation of experimental tree-dimensional (3D) structural information for many organic molecules of biological interest and the development of fast and accurate methods for the 3D structure generation for chemical molecules have led to the development of 3D structural descriptors. 3D QSAR involves the analysis of quantitative relationship between the biological activity of a set of molecular compounds and their three-dimensional structures [100].

Figure 5.2 shows the structure for a typical QSAR data set. Here, **X** and **Y**-data represent independent descriptive features (or descriptors) and the corresponding interests that need to be predicted. The indexes ***i***, ***j***, and ***k*** denote the molecule names (ID), biological responses, and features, respectively. The aim of the QSAR studies is to find a model such that the model predicts the relationship between the independent **X**-data and the dependent **Y**-data for future values of **X**-data with unknown **Y**-data.

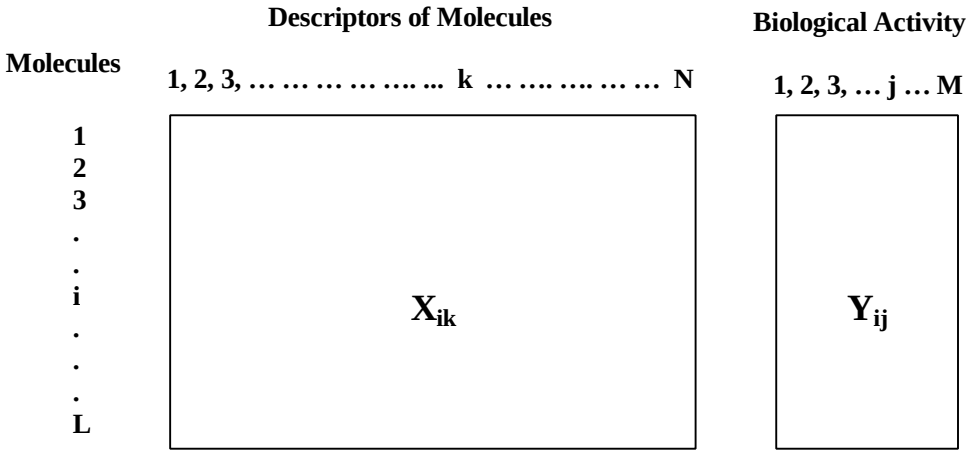


Figure 5.2 A Typical QSAR Dataset [99]

Often QSAR problems have more descriptive features than there are compounds (or molecules). If the number of features becomes larger than the number of molecules, the predictive modeling problem becomes extremely challenging.

Recent trends in both 2D and 3D QSAR studies have concentrated on the development of optimal QSAR models through feature selection [100]. The objective is to narrow a large set of potential inputs into a smaller subset for good prediction. The selection of a subset of features is very important because an excess number of features in

the model cause two problems [92]. The first problem relates to the fact that excess features cause the parameter estimation methodology to overfit the model so there is no real predictive value. The second problem is that excess features might make the model more difficult to explain.

5.3 Feature Selection for In-Silico Drug Design

Variable selection is typically a time-consuming and ambiguous process for QSAR because of the large number of descriptors in the datasets. Most 2D and 3D QSAR techniques assume a linear relationship between the biological activity and molecular descriptors (or features). The optimal set of descriptive features is typically selected by combining stochastic search methods with the correlation methods such as Multiple Linear Regression (MLR), Principal Component Analysis (PCA), and more commonly Partial Least Square (PLS) regression [99, 101-103]. PLS regression is a principal component analysis related regression method. In PLS regression, a relationship is determined between a response variable $\boldsymbol{Y}$ and feature data matrix $\boldsymbol{X}$. Latent variables are determined in such a way that they model the feature data set $\boldsymbol{X}$ and optimally correlate with response variable $\boldsymbol{Y}$ [104, 105]. In contrast, in principal component analysis the latent variables (eigenvectors) only transform feature data set $\boldsymbol{X}$ in an orthogonal set. Some feature selection techniques in PLS analysis have been proposed such as the GOLPE [106], VIP [107], and IVS [108] methods.

i) The GOLPE (**G**eneration of **O**ptimal **L**inear PLS **E**stimators) uses D-optimal design to preselect non-redundant variables and fractional factorial design to run

PLS analyses with different variable combinations [106, 109]. Variables significantly contributing to prediction are then selected.

ii) VIP (**V**ariable **I**nfluence on the **P**rojection) score is derived from the PLS weights for each variable [107]. VIP scores greater than 1.0 indicate important variables. The threshold value of 0.8 is used as a limit below which variables are considered to be unimportant.

iii) IVS (**I**nteractive **V**ariable **S**election) method uses cross-validation for dimensionwise elimination of single elements in the PLS weight vectors [108, 110-112].

In recent years, a variety of nonlinear QSAR methods have been proposed. Most of these methods are based on Artificial Neural Networks (ANNs) [113, 114] or other related machine learning techniques such as Support Vector Machines. Simulated annealing, genetic algorithms [115-120], and evolutionary algorithms [92, 121, 122] have been used as stochastic search methods for feature selection, which are then wrapped around the available linear or non-linear QSAR methods.

Waller and Bradley [122] have proposed a **F**ast **R**andom **E**limination of **D**escriptors (FRED) algorithm. FRED is a simple random selection strategy that implements an iterative generation of models. It starts with a population of predictive models composed of a fixed or variable number of selected descriptors and eliminates descriptors iteratively. After creating a population of models, the models are sorted according to their fitness. This fitness function is based on Partially Least Square (PLS) regression model with full (leave-one-out) cross validation. A certain number of models with low fitness values are deleted from the population. The descriptors of the deleted

models (or less fit models) are then compared to the descriptors of the models kept in the population (more fit models). Descriptors that are not found in the more fit models are placed in the TABU list. If a descriptor appears on the TABU list more than once (not necessarily sequential generation), it is removed from the allowable descriptor pool. A new population of models is created for next generation by using those descriptors in the allowable pool. Iterative elimination of descriptors leads to subsequent generations consisting of more fit models.

Kewley, Embrechts, and Breneman [92] have proposed a novel Artificial Neural Network (ANN) based technique for Data Strip Mining (DSM), which results in a predictive model for data sets with a large number of potential input features and comparatively few data points. DSM uses neural network sensitivity analysis to iteratively eliminate variables, which are less significant in the model. After an ANN has been trained on a relatively large set of input features, the significance of each feature is calculated by holding all the input features frozen at their average value and tweaking the features one at a time and identifying the most sensitive input features that cause the largest output variability.

Genetic algorithms [117, 118] have been used extensively for feature selection in QSAR. Most of these studies adopted classical binary string representation. Each individual in a population represents a binary string of digits, either “one” or “zero”. The values of “one” or “zero” imply that the corresponding descriptive feature is included or excluded in the parent (model). The length of the chromosome string is equal to the total number of descriptors (all descriptors). The population is evolved with a GA by

performing classical crossover and mutation operators. The fitness of each individuals is evaluated with the PLS algorithm with a leave-one-out cross validation procedure.

Wessel, et al. employed a GA coupled with computational neural networks for feature selection in order to predict the absorption of a drug compound through the human intestinal cell lining [123]. A predictive ANN model is developed by using GAs with a neural network fitness evaluator. They used a set of 86 drug and drug-like compounds and their experimentally derived Human Intestinal Absorption rates (%HIA), which were gathered from literature sources. Each compound is represented by 728 descriptors. They employed data preprocessing methods (called objective feature selection [124]) in order to eliminate features that contained redundant or minimal information. The first objective feature selection method was to eliminate descriptive features that had greater than 80% identical values. The second one is to eliminate one of the features, which were correlated higher than 90%. 127 descriptive features remained after objective feature selection method. The 127-member reduced pool of descriptive features was fed into GA/ANN in order to build a good predictive model with a few descriptive features. The data set was split randomly into three sets: training set with 67 molecules, validation set with 9 molecules, and external prediction set with 10 molecules. The root mean square errors (RMSE) of validation and test sets were used by the GA to determine a fitness function that related directly to the overall quality of an individual (a particular subset) in the population. The fitness function is calculated with the following equation:

$$\text{Fitness} = T_{RMSE} + W\,(\,|\,T_{RMSE} - V_{RMSE}\,|\,)$$

Where T_{RMSE} is the training set RMSE, V_{RMSE} is the validation set RMSE, and W is the weight factor. They used 0.4 for weight factor W based on their experiences.

5.4 Feature Selection in Statistics

Selecting a subset of variables (features) is a problem that has been extensively considered in linear models. The variable selection problem is considered as a special case of the model selection problem, where each model corresponds to a distinct subset of variables. In order to select a model containing subset of variables, several criteria have been proposed in the statistical literature for linear models. One of the well-known criteria is the Residual Mean Square (RMS) of prediction, which is defined for a model with **p** variables as

$$RMSp = \frac{SSEp}{n - p}$$

where SSE is the residual sum of square errors, and **n** is the number of data points. If two models are compared; then, the one with the smallest RMSp is chosen.

The other well-known subset selection criterion for linear regression is Mallow's Cp statistic [125]. Since predicted values obtained from a subset regression model are biased, the mean square error of prediction consists of two components: the variance of prediction arising from estimation, and a bias arising from the deletion of variables [126]. Mallow's Cp statistics is defined as

$$Cp = \frac{SSEp}{\hat{\sigma}^2} + (2p - n)$$

where $\hat{s}$ is the estimate of the variance of the random error, which calculated from the full regression model. The expected value of the **Cp** is **p** when there is no bias in the

model. Therefore, the feature subset whose **Cp** is the closest to **p** is the best subset based on **Cp** statistic.

When the number of variables is large, the evaluation of the all possible subsets of variables using one of the above selection criteria is not feasible. There are sequential variable selection methods that do not require evaluating all possible subsets [127, 128]. These methods calculate the variation caused by the deletion or the addition of a variable into the model. They keep the variable whose contribution into the model is significant based on a selection criterion. Those methods are forward selection, backward elimination, and stepwise regression.

The sequential variable selection methods do not require evaluation of all 2^k-1 possible subset models, where k is the number of the independent variables. These methods involve evaluation of at most (k+1) models. They are deterministic methods in the sense that they always obtain same subset of features for all runs. However, the order of inclusion or deletion of variables should not be interpreted as reflecting the relative importance of the variables [126]. These methods generally select the same variables from datasets with noncollinear variables. They do not perform well on collinear datasets. Ridge regression can be used to select variables when data are highly collinear. Ridge regression is a method to stabilize the regression coefficient [126], where a stable coefficient means that it is not affected by slight change made in data. Several Bayesian methods for variable selection in Multiple Linear Regression have been proposed. Most of them use either the Akaike Information Criterion (AIC) [129] or the Bayesian Information Criterion (BIC) [130] for model selection. A detailed review of feature (variable) selection in Multiple Regression is given by Thompson [127, 128].

5.5 Common Components of Feature Selection Algorithms

A feature selection algorithm performs a search through the space of feature subsets and must have at least the following three components [131]:

- A feature evaluation criterion to compare variable subsets.
- A search method to explore the possible variable combinations of the search space.
- A stopping criterion to stop searching through the space of feature subsets.

5.5.1 Feature Evaluation

The feature evolution criterion helps us to compare subsets of features in order to select one of them. Feature selection methods, which perform feature selection as a preprocessing step prior to learning, can be divided into two main categories based upon their feature subset evaluation: a ***wrapper*** or a ***filter*** approach [6]. A wrapper method searches for a good feature subset tailored to a particular predictive modeling approach and uses the induction algorithm as a black box for evaluating feature subsets. On the other hand, a filter method attempts to access the merits of features from the data alone.

Wrapper methods generally use the actual learning algorithm with a statistical re-sampling method (such as cross validation, bootstrapping) to estimate the quality of feature subsets. This approach is useful but sometimes computationally prohibitive because the learning algorithm to evaluate feature subsets has to be applied numerous times. On the other hand, filter methods have proven to be more practical than wrapper methods for applications to large datasets since they are much faster.

Figure 5.3 Filter Approach to Feature Selection

5.5.2 Search Methods

The only way to be sure of selecting the best subset of variables would be to enumerate all the possible models. Given the fact that if **n** is the number of variables; then there are ($\mathbf{2^n-1}$) possible combinations. This approach is limited to problems with a few variables. Searching the whole space within reasonable time is necessary if a feature selection algorithm is to operate on data with a large number of features.

Greedy Hill Climbing (GHC) methods consider local changes to the current feature subset by adding or deleting a single feature from it [95, 132]. If a GHC considers only additions of features to the feature subset it is called as forward selection. Considering only deletions is called as backward elimination. The combination of these two methods, called stepwise bi-directional search, considers both addition and deletion of features. Although GHC methods are fast, they do not enable the user to consider several contending subsets involving different sets of variables that may be preferable to the selected one for other reasons.

GAs have proven to be a very effective techniques to find a solution for feature selection. They employ a population of potential solutions, which search the solution space in parallel. GAs allow the exploration of the whole search space and generally select more efficient subsets of features than that of classical feature selection methods [7].

5.5.3 Stopping Criterion

If we would have been able to enumerate all possible subsets, we could have selected the best one according to our evaluation criterion. Here, the feature selection algorithm is terminated when all subsets are evaluated.

If the empirical distribution of the evaluation criterion is known, a statistical hypothesis testing can be applied to terminate the search. The distribution characteristics of evaluation criteria in classical methods can be easily derived. For example, sequential variable selection methods in Multiple Linear Regression models terminate as soon as possible when a variable is found insignificant according to the statistical test.

The distribution of the evaluation criterion is difficult to obtain for non-parametric methods such as Artificial Neural Networks. Therefore, hypothesis testing is seldom used with these models. The most frequently used method is to compute the estimation of the generalization error on validation set using a bootstrapping or a cross-validation methodology. The variable subset with the best predictive performance will be selected. GA based feature selection methods assume a fast and efficient learning method (e.g. PLS). Furthermore, these methods may become computationally prohibitive if a user wants a high level of statistical confidence.

5.6 GAs for Feature Selection

5.6.1 Representation

The most important part of applying GAs to a specific problem is the selection of a suitable genetic coding (representation) and fitness function for individuals. The

representation must cover the whole search space for the problem. The natural and simplest representation for the feature selection is a binary string representation. The number of features in the problem determines the length of the string (chromosome). Each bit (gene) represents the elimination or inclusion of the associated feature. As an example, consider a problem with six features. The string 100001 corresponds to selecting features 1 and 6. The advantage of binary representation is that classical GA operators (binary mutation and crossover) can be applied without any modification.

5.6.2 Studies of GAs on Feature Selection

Several researchers have employed GAs for feature selection in classification problems [89, 133]. In these studies, individuals were encoded as standard binary strings and a learning algorithm (classifier) was used to calculate the fitness.

Guerra-Salcedo and Whitley [89] compared two different genetic algorithms for feature selection in a classification problem. They used the Euclidian Decision Table classifier as a fitness function. One of the first general purpose GAs is called GENESIS [134] and is a public domain software based on a simple GA. A more advance general purpose GA is CHC proposed by Eshelman [135]. CHC is a generational search algorithm. They reported that CHC performed better than GENESIS.

The CHC algorithm randomly pairs individuals in the population. Only pairs, which differ from each other by some number of bits (mating threshold), are crossed over. This is called a truncation selection scheme. The crossover operator is similar to uniform crossover and randomly swaps exactly half of the bits that differ between two strings. When no offspring is produced, the mating threshold is reduced by 1. When the

mating threshold become 0 and no offspring can be created, a new population is created based on the best individual in the current population. This is called cataclysmic mutation. Cataclysmic mutation uses the best individual of the current population as a template to create a new population by mutating the template.

Leardi, Boggia, and Terrile [7] applied a modified simple GA to regression problems. They used the Multiple Linear Regression as a fitness function and cross-validated variance explained by the regression as a fitness. In order to prevent an infeasible individual (in which there are more variable than objects in regression) feasible individuals are created for the initial population. Crossover is performed in the following way after selecting two donors, each gene has swapped with a certain probability. Immediately after the creation of two offspring their response is evaluated and a decision is made whether these offspring will be inserted to current population or not. If an offspring with ***k*** variable has higher fitness than that of the worst individual with ***k*** variable in the current population it will be replaced. Otherwise, it will be discarded. The mutation operator randomly chooses some bits and changes their value from 0 to 1and vice versa. After all mutation have been performed the fitness of the mutated individuals are evaluated and decision is made whether they are inserted into the current population or not. This GA is successfully applied to regression problems and performed better than classical variable selection methods.

Hou, Wang, Liao, and Xu [115] studied QSARs for 35 cinnamamides by using genetic algorithms. They used binary string representation and classical crossover and mutation operators. The probability of an individual to be parent for crossover was based on fitness of that individual. The fitness function was defined as the multiple linear

regression coefficient (***r***). After the crossover and mutation operator, the offspring are compared with the individuals in the population. Offspring that are better than some individuals in the population are copied to the population. A partial re-initialization is applied after some number of genetic operators. In partial re-initialization lowest 50 percent of chromosomes are replaced with randomly generated ones.

Yasri and Hartsough [136] reported the development of a novel QSAR technique combining genetic algorithms and neural networks for selecting a subset of relevant features and building the optimal neural network architecture for QSAR studies. The optimal neural network architecture is explored in parallel with the feature selection by dynamically modifying the size of the hidden layer. They showed that this method could be used to build both classification and regression models and outperformed simpler feature selection methods mainly for nonlinear datasets. In their proposed algorithm, each individual (chromosome) encodes different subset of features by a binary string representation. The length of each chromosome is equal to the total number of features in the data set. A neural network is trained and cross validated for each chromosome using a training dataset. The classical crossover mutation and roulette wheel selection were used to evolve the GA population.

CHAPTER 6

Data Strip Mining

Feature reduction is an important step for building good predictive models. Too many features in the model cause the classical curse of dimensionality problem. Furthermore, irrelevant or redundant features not only diminish the performance of a predictive model but also make the model more difficult to explain and interpret. Data strip mining was introduced as an iterative procedure for feature reduction/model building for data sets where the number of features exceeds or is on the order of the number of data records [92]. Figure 6.1 depicts the process of data strip mining.

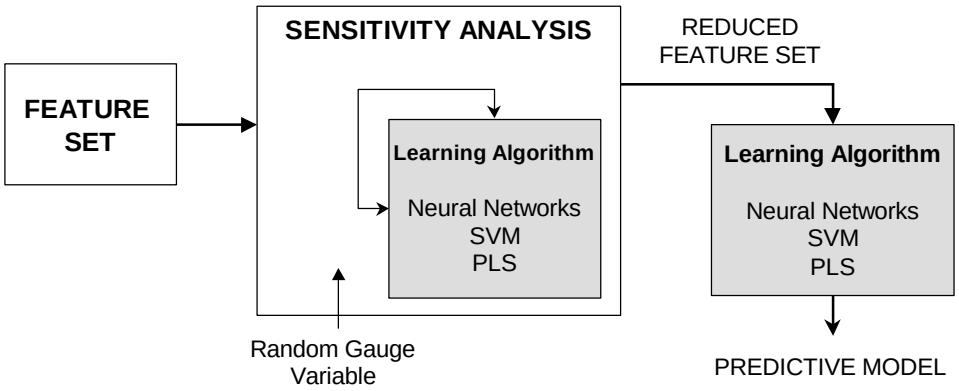


Figure 6.1 Data Strip Mining Process

6.1 Scientific Data Mining

Data mining is the process to automate the discovery of non-obvious, novel, and potentially useful information from large datasets with the aid of computers. Scientific data mining refers to data mining that assists the scientific discovery process or is applied to science-related data. Scientific data mining differs from commercially-oriented data mining with regard to the type of data (both in size and data quality) and uses different methods as well. While the boundary between traditional data mining and scientific data mining is not clearly delineated, scientific data mining applications more often rely on real-number coded data and frequently require regression rather than classification. Data mining software developed for traditional data mining applications is often not well suited for scientific data mining applications.

The scientific data mining process can often be broadly divided into three steps:

i) The prediction of future unknown values of one or more dependent (response) variables in a data set, by using some or all of the other descriptive variables (features) in a predictive model;

ii) Feature selection for identifying the most relevant set of descriptive variables;

iii) The model explanation phase, where the selected features or combinations of selected features explain part of (or the complete) model in a way that can be understood by the domain expert.

The data mining process forces scientific discovery to proceed in a systematic way and assists in discovery by revealing hidden, non-obvious patterns and information that is ultimately interpretable, explainable, and verifiable by humans.

6.2 StripMiner™

The StripMiner™ [4] code is a software package for predictive modeling and feature selection with scientific data mining applications in mind. StripMiner™ is a shell program for data-strip mining that manages and integrates the execution of several different machine learning and statistical methods such as Artificial Neural Networks (ANNs), Genetic Algorithms (GAs), Support Vector Machines (SVMs), Local Learning (LL), and Partially Least Square (PLS) Regression. StripMiner™ also implements several different methods for feature reduction for predictive modeling. These feature reduction methods are based either on genetic algorithms or sensitivity analysis. Feature reduction methods specific to the particular predictive model are also possible.

6.3 Predictive Modeling Algorithms in StripMiner™

The strip-mining approach does not depend on any particular methodology. There are several machine learning and statistical methods for building predictive models. StripMiner™ gives the user the choice between several machine learning approaches such as ANNs [92], SVMs [137], GAs [22] [8], and LL [138] as well as a statistical method, PLS [139].

6.3.1 Neural Network Model

The neural networks models are standard feed-forward multi-layered perceptrons (MLP) trained with the backpropagation algorithm [140, 141]. The neural networks have two hidden layers, are oversized (e.g. 13 and 11 neurons in the hidden layers). Training was halted with early stopping by monitoring the validation set. Because of the early

stopping procedure, the neural network results are not very sensitive to the number of neurons in the hidden layers. On the other hand, neural networks that rely on early stopping tend to be more linear.

6.3.2 Local Learning

Local Learning is a data analytic modeling approach that attempts to model the training data by only fitting a parametric function in a region around the location of a query point [142-144]. This means that local learning methods are *locally parametric*, as opposed to most learning methods (Least Square Regression, etc.) that attempt to fit a single global model into the training data. Local learning methods make predictions based on local models constructed on the neighborhood of a query point. Nearest neighbor and locally weighted regression are examples to local learning models [142]. In Chapter 10, genetic algorithms with local learning are developed for feature selection problem.

6.3.3 Support Vector Machines

StripMiner^{TM} uses standard 2-*Norm* Support Vectors Machines (SVMs) for regression problems [137]. The detailed tutorial information about SVMs can be found in [145]. A short introduction to SVM regression taken from [146] is given below.

Suppose we are given t training examples (x_i, y_i), where, $i = 1,2,\ldots,t$, $x_i \in \mathbf{R}^n$, and $y_i \in \mathbf{R}$. The optimal ε-insensitive SVM model can be found by solving the following dual quadratic problem:

$$\begin{aligned} &\text{minimize } \frac{1}{2}(\alpha^* - \alpha)^T K(\alpha^* - \alpha) - (\alpha^* - \alpha)^T y + \varepsilon(\alpha^* + \alpha)^T \mathbf{1} \\ &\text{s.t.} (\alpha^* - \alpha)^T \mathbf{1} = 0, \;\; 0 \le \alpha_i \le C, \;\; i = 1,...,t \end{aligned} \tag{6.1}$$

where K is a matrix with $K_{ij} = k(x_i, y_i)$ and **1** is a vector of ones. The optimal regression function is obtained by

$$\hat{y} = f(x) = \sum_{i=1}^{t} (\alpha_i^* - \alpha_i)^T k(x_i, x) + b \tag{6.2}$$

There are several types of kernels (e.g. linear, polynomial, and radial-basis function) that could be used in the SVMs. The SVM in this dissertation uses Radial Basis Functions (RBF) kernel with parameter σ, which is the most common used kernel in the SVM literature:

$$k(x_i, x) = \exp\left(-\frac{\|x_i - x\|^2}{2\sigma^2} \right) \tag{6.3}$$

Typically, SVM algorithms with RBF kernel function perform better with normalized data [147]. SVMTorch is used as a subroutine to solve equation (6.1) [148]. The solution of equation (6.1) requires the following priori knowledge: the kernel parameter σ, the trade-off constant C, and the value of ε. These parameters are automatically computed using a Pattern Search (PS) algorithm [146, 149]. PS is a direct search method that does not use derivatives but direct function evaluation to find optimal points. For solving regression problems with good accuracy, it is necessary to choose a sufficiently good parameter set for the SVM. Automatic model selection is an important issue for SVM and is a hot research topic today [150]. The leave-one-out-error (LOO) estimate is often used

for the classification. However, automatic model selection for regression has not been studied as much as that for classification. In StripMiner^{TM}, the Q^2 averaged over the leave-one-out cross validation is used as a measure of validating the predictive model and optimized by using the pattern search algorithm [149]. The pattern search methods are a kind of direct search method. They do not use any derivatives but use direct evaluation of points in the search space. Therefore, they can be applied for problems that are impossible or for hard to obtain information about the derivatives.

6.3.4 Partial Least Square (PLS) Regression

Partial Least Squares (PLS) regression is an algorithm similar in idea to principal component analysis (PCA) that has found utility in solving a variety of data analysis problems [104, 151, 152]. The Partial Least Squares method seeks to uncover a small number of "latent" variables from a much larger set of correlated descriptors. The PLS method can be expressed as:

$$y = a_1 LV_1 + a_2 LV_2 + \ldots\ldots + a_m LV_m$$

in which y is the dependent variable (biological response), LV_i the i^{th} latent variable and a_i is the i^{th} regression coefficient corresponding to LV_i . Each latent variable, LV_i, can be expressed as a linear combination of independent variables x_i:

$$LV_i = b_1 x_1 + b_2 x_2 + b_3 x_3 + \ldots. + b_n x_n$$

where x_i is the i^{th} independent molecular descriptor and b_1, $b_2 \ldots b_n$ are the descriptor coefficients. The first latent variable accounts for most of the variance, while consecutive latent variables account for relatively smaller amounts of variance. In addition, the latent variables in a model are orthogonal to each other.

6.4 Performance Estimation for Learning Models

In this dissertation, the model quality for a machine learning model is measured with five distinct measures for the error: the root mean square error (**RMSE**), $\mathbf{r^2}$, $\mathbf{q^2}$, $\mathbf{R^2}$ and $\mathbf{Q^2}$. In this dissertation, qualities of the fit of the training data (goodness of the fit of the model) are always reported in terms of $\mathbf{r^2}$ and $\mathbf{R^2}$; and test (validation) errors are always reported in terms of **RMSE**, $\mathbf{q^2}$ and $\mathbf{Q^2}$. In the given formula below, summations are inclusive of all compounds (data points) in the training or test set.

$\mathbf{r^2}$ is known as the coefficient of determination in which, **r** is the coefficient of correlation (Pearson's correlation coefficient) that is a measure of the degree of linear association between actual activities (y_i) and predicted activities ($\hat{y}_i$). **r** is given by

$$r = \frac{\sum (y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum (y_i - \bar{y})^2 \sum (\hat{y}_i - \bar{\hat{y}})^2}},$$

where $\bar{y}$ is the mean of the actual (response variables) activities and $\bar{\hat{y}}$ is the mean of the predicted (response variables) activities. $\mathbf{q^2}$ is defined for a test set as $q^2 = 1 - r^2$.

$\mathbf{R^2}$ is defined for a training set by

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

$\mathbf{Q^2}$ is defined for a test set by $Q^2 = 1 - R^2$, which is given by

$$Q^2 = \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

In the QSAR literature, the prediction quality is often reported via $\mathbf{q^2}$ or $\mathbf{Q^2}$, which is called as ***predictive*** $\mathbf{R^2}$ or ***cross-validated*** $\mathbf{R^2}$. It is calculated in the same way as in $\mathbf{R^2}$ but the predicted activities ($\hat{y}_i$) are always obtained from the leave-one-out-cross-validation. In this dissertation, $\mathbf{q^2}$ and $\mathbf{Q^2}$ are always calculated on the test or validation sets as explained above and are different than the ***predictive*** $\mathbf{R^2}$ (***cross-validated*** $\mathbf{R^2}$) used in the QSAR literature.

The **RMSE** is the square root of the average of squares of the errors for each of the N data points of the test set according to

$$\text{RMSE} = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{N}},$$

where **N** is the number of data points in the test set and summation is inclusive of all data points in the test set..

6.5 Sensitivity Analysis for Feature Reduction

The purpose of sensitivity analysis is to determine the saliency of each of the features in a model and to reduce the number of features for the model. Sensitivity analysis uses a trained machine-learning model to determine the sensitivities of the variables of the model [3, 92, 153, 154]. For more details about sensitivity analysis, see Arciniegas [155].

6.5.1 One-Dimensional (1-D) Sensitivity Analysis

In order to proceed with 1-D sensitivity analysis, all the descriptive features are held frozen at their median (or average) values and the variation in the output of the learning model is monitored, while the features are changed one at a time within their allowable range. In practice, rather than changing a feature continuously, the model output for each feature is monitored for certain number of different discrete values (e.g. 13 level). The sensitivity for a particular feature is the maximum model response minus the minimum model response from these 13 settings.

The feature set was extended with an additional random variable to gauge the sensitivities. Although the random variables used in this dissertation were obtained from a uniform distribution, they can be drawn from different distributions. Features with sensitivities smaller than this random variable are eliminated in successive (strip mining) iterations for feature reduction stages, and a new model based on the reduced feature set is constructed. Iterative feature elimination with sensitivity analysis was halted when there were no more features with sensitivity below the sensitivity of the random gauge variable. Here, the hypothesis is that features with sensitivities that are lower than this random variable are not important for the model.

Figure 6.2 shows graphically how sensitivity analysis works. The roman numerals show the steps in the sensitivity analysis. In step I, a random variable is added into the training dataset. The second step (step II) in this feature reduction methodology is the training of a machine-learning model based on the extended dataset -the original dataset augmented by a uniform random variable. So far, only one uniform random variable was

considered as a gauge. It is possible to add several random gauge variables (possibly from different distributions) at the same time.

In order to implement the sensitivity analysis, a new ***sensitivity analysis specific dataset*** is generated in the following way. All the features were held at their median value and each feature was changed one at a time between allowable ranges of the feature for L discrete levels. Here, all features are scaled between zero and unity. The structure of the sensitivity analysis specific dataset for the sensitivity analysis is shown in Figure 6.2. It is assumed that there are M descriptive features, N data points, and L sensitivity levels for one dependent (response) variable.

In step III, the sensitivity analysis specific file is fitted into the trained machine-learning model. The generated model response is an array, consisting of M sets (because there are M features) of L predicted (output) values. In step IV, the sensitivity for a particular feature is then estimated as its range (i.e., maximum minus the minimum model response) from the L level settings for that feature.

Finally in step V, all features with sensitivities smaller than the random variable are dropped and one cycle of the sensitivity analysis ends. A new feature reduction cycle starts with the dataset with selected features and the random variable. This iterative feature elimination procedure is halted when no further features can be dropped (i.e., there are no more features with a sensitivity below the sensitivity of the random gauge variable) or when a predefined number of features has been reached.

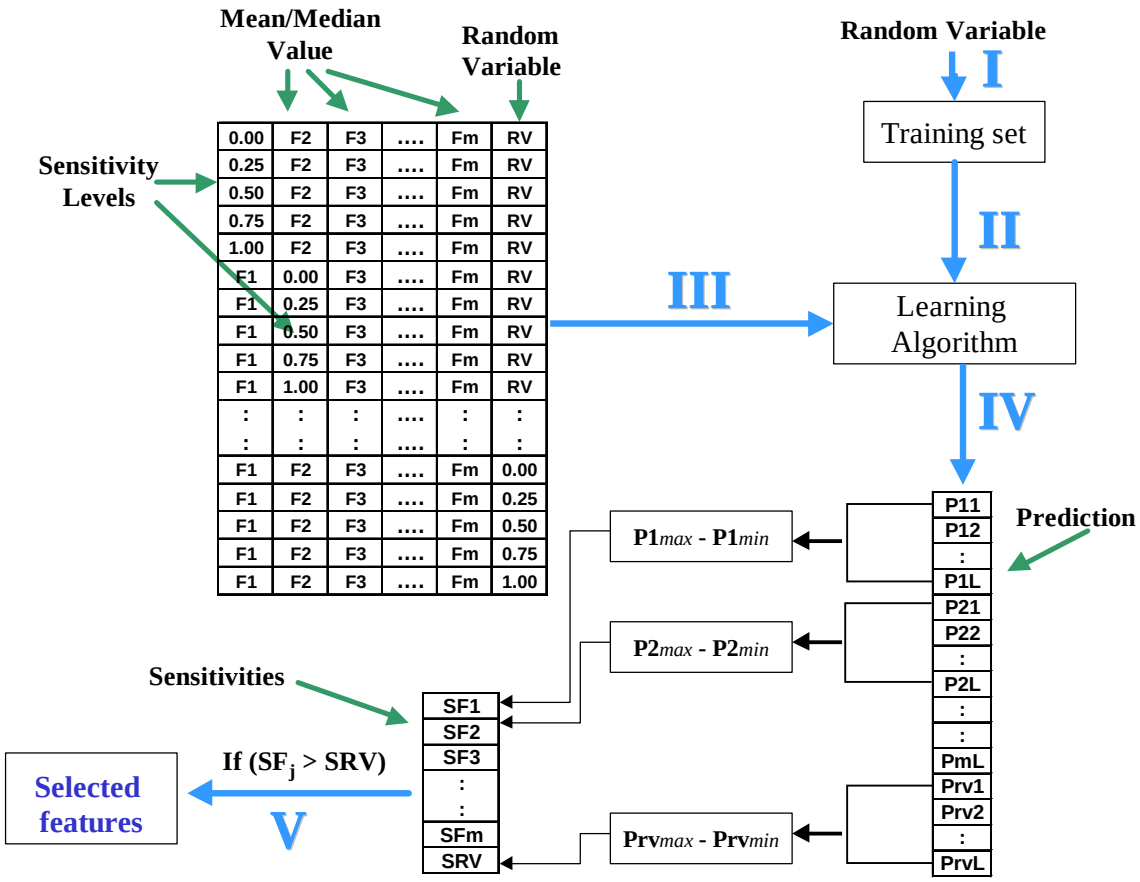


Figure 6.2 One Cycle in the Sensitivity Analysis

1-D sensitivity analysis can be further extended to multivariate sensitivity analysis (e.g. 2-D, 3-D). For example, 2-D sensitivity analysis is similar to 1-D sensitivity analysis, except that saliency is calculated for a pair of features by changing two features simultaneously while holding the other features frozen at their average or median values.

6.5.2 Bootstrap Aggregation (Bagging) Sensitivity Analysis

Bootstrapping, introduced by Efron [156], is a well-known technique for estimating the generalization error of a learning model based on resampling. In classical bootstrapping, a sample of data of size ***n*** is taken uniformly from the original data of size

n with replacement and this resampling process is repeated in a certain number of times (50-1000) [157]. In StripMiner™, the (***n-v***) samples for the learning set are drawn without replacement, where ***n*** is the number of samples in the data set and ***v*** is the number of samples in the validation set. There is no overlapping between the training and validation sets for each bootstrap sample.

The machine learning model is trained by using the training set and the sensitivities of the variables are measured. The validation data set is fitted into the trained learning algorithm in order to assess the predictive ability (goodness) of the model. This process is repeated for a number of different bootstrap samples. This goodness measurement can be used to weigh (or bag) the sensitivities of the features over assemble of models.

The final sensitivity for each variable is obtained by sorting the weighted average of all bootstrapping learning models sensitivities in ascending order, and by eliminating any features with sensitivities smaller than that of the random variable. This elimination process proceeds in successive stages using the reduced dataset for the next stage for further possible feature (descriptor) reduction. This process is terminated when no more features can be dropped (i.e. all features are less sensitive than the random variable) or when a predefined number of features have been reached. Figure 6.3 graphically depicts the bagging sensitivity analysis. The roman numerals in Figure 6.3 represent steps in the bagging sensitivity analysis.

In step I, the full dataset is extended with a random variable. In step II, a bootstrap sample with size ***l*** from the full dataset with size ***n*** is drawn without replacement to construct the training (learning) set. The remaining set with size ***v*** = ***n*** - ***l*** constitutes the

validation set. In step III a machine-learning algorithm is trained by using the training set. In step IV, the sensitivity analysis specific file is fitted into the trained learning model. The generated model response is an array consisting of M sets (because there are M features) of L predicted (output) values. In step V, the sensitivity for a particular feature is then estimated as its range (i.e., maximum minus the minimum model response) from the L level settings for that feature. In step VI, the validation set is fitted into the trained learning model. The quality of prediction is measured in terms of $\mathbf{q^2}$, where $q^2 = 1 - r^2$. Here, r is the correlation coefficient between actual response and predicted response. In step VII, the sensitivities of features are weighted by multiplying each feature by $1 - q^2$. Steps I through VII are repeated for a certain number of bootstrap samples. Weighted sensitivities of features for each bootstrap sample are recorded. Finally in step VIII, the final sensitivity for each feature is calculated as the weighted average of all bootstrapping sensitivities. All features with weighted average sensitivities higher than the weighted average sensitivity of the random variable are kept, and one cycle of the bagging sensitivity analysis ends. A new bagged feature reduction cycle starts with the dataset with selected features and the random variable. This iterative feature elimination procedure stops either when no further features can be dropped or when a predefined number of features has been reached.

The data strip mining procedure is independent from a particular modeling approach and can be integrated with any machine learning or statistical modeling approach in general. Sensitivity analysis for feature reduction proceeded with predictive models based on MLP neural networks.

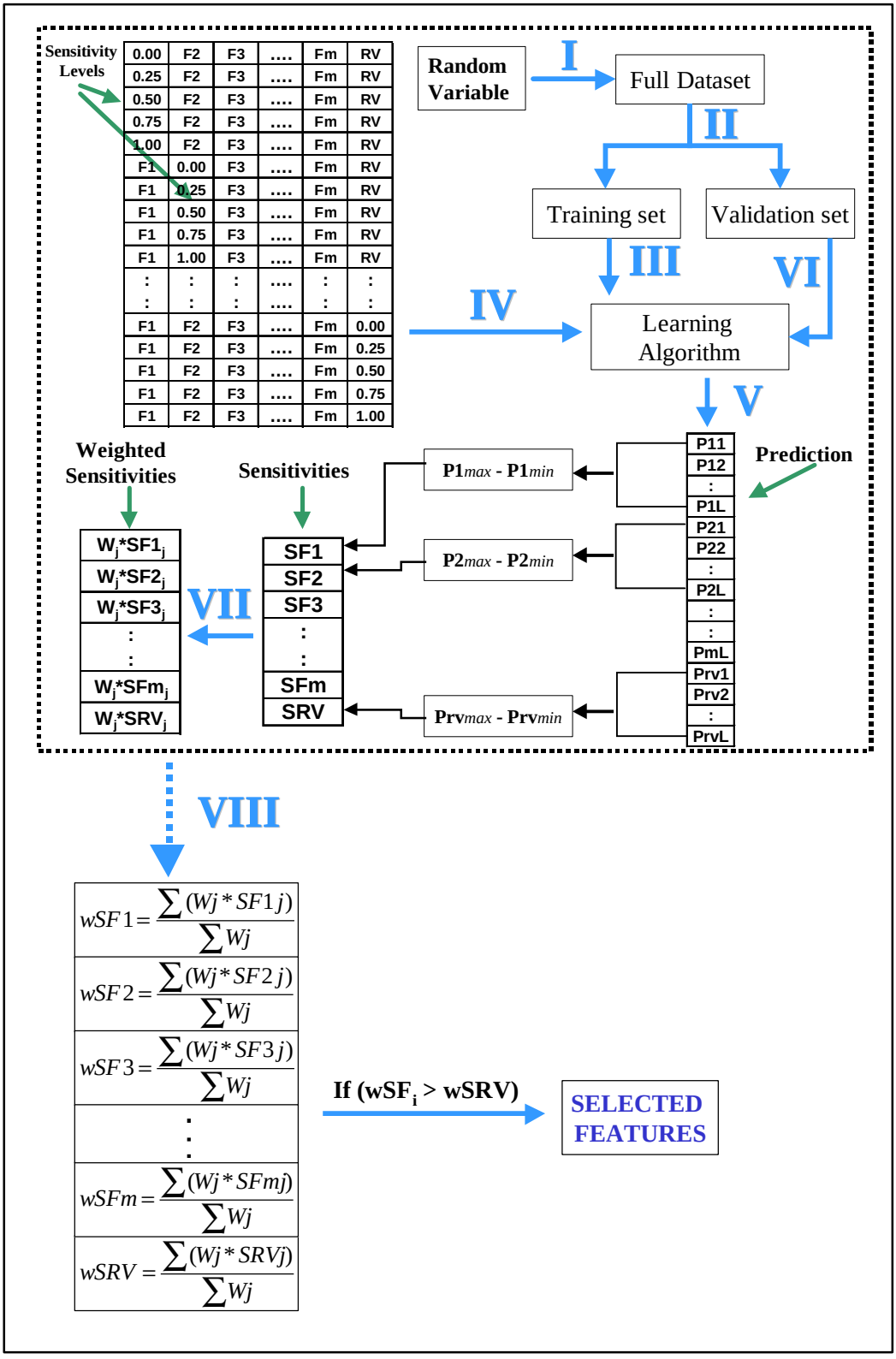


Figure 6.3 Bagging Sensitivity Analysis

CHAPTER 7

Benchmarking Datasets

In this chapter we introduce the QSAR datasets, which will be used for benchmarking the proposed algorithms. Since the aim of QSAR is to predict the biological activity of new untested molecules from the knowledge of their chemical properties, descriptors are needed to be generated. We first give a brief description about the descriptors that were generated for the benchmarking datasets and then, three benchmarking datasets (Lombardo, HIV_{rt}, and Caco2) will be introduced.

7.1 Descriptors

QSAR assumes that the change in biological activity that is observed within a series of similar compounds is a function of the change in chemical structure within the series [99]. Thus, QSAR methods deal with identifying important structural features of molecules that are relevant to explain variations in biological or chemical properties. The QSAR datasets used in this dissertation consist of either Transferable Atomic Equivalent (TAE) descriptors [158, 159], or Property Encoded Surface Translator (PEST) descriptors [160, 161], or Molecular Operating Environment (MOE) descriptors [162], or combination of those descriptors. These descriptors are briefly introduced in the following sections.

7.1.1 Transferable Atomic Equivalent (TAE) Descriptors

The TAE method employs rapid reconstruction of charge densities and electronic properties of molecules, using atom charge density fragments that are pre-computed from *ab initio* wave functions [158, 159]. The original TAE descriptors were created from ten or twenty-bin fixed-range histograms describing the distributions of ten different electronic properties on electron density-derived Van der Waals molecular surfaces. A new class of Wavelet Coefficient Descriptors (WCDs) has been developed that encodes molecular surface property information into a small set of wavelet coefficients [163]. Development of rapidly calculable WCDs captures important features of molecular electron density distributions.

7.1.2 Property Encoded Surface Translator (PEST) Descriptors

In Shape/Property-based PEST (Property Encoded Surface Translator) descriptors a TAE property-encoded surface is subjected to internal ray reflection analysis [160, 161]. In this method, a ray is initialized with a random location and direction within the molecular surface, and then reflected throughout inside the electron density isosurface until the molecular surface is adequately sampled. Molecular shape information is obtained by recording the ray-path information, including segment lengths, reflection angles and property values at each point of incidence.

7.1.3 Molecular Operating Environment (MOE) Descriptors

The Molecular Operating Environment (MOE) is a flexible and robust chemical computing software by the Chemical Computing Group [162]. MOE can calculate over

300 molecular properties including topological indices, octanol/water logP, molar refractivity and property-encoded Van der Waals surface descriptors, which have shown wide applicability in compound classification, QSAR and ADME property modeling [162].

7.2 Lombardo Blood-brain Barrier Dataset

The Lombardo blood-brain barrier partitioning data set is a well-known and challenging dataset for QSAR benchmark studies [164]. Sixty-two molecules from the original data set were chosen for this study, with 694 features. Molecule 37, butanone, was excluded from our study since an experimental value was not reported. The Lombardo data set contains two very distinct classes of molecules, a class comprised of 36 nitrogen-containing heterocycles and a class of 28 alkanes, alkenes, and halogen-substituted derivatives. In addition, the gases methane and nitrogen are included.

The Lombardo dataset contains a diverse set of molecules for which blood-brain barrier penetration data is available. This is an important phenomenon to model, since all drugs that act on the central nervous system must pass through this barrier in order to function. This class of compounds includes antidepressants, anesthetics, chemotherapy agents, antifungal agents, antibiotics and antiviral drugs. The Lombardo data set represents a benchmark in QSAR and provides a wealth of comparison data in QSAR literature.

7.3 Human Immunodeficiency Virus reverse transcriptase (HIV_{rt}) Inhibitors Dataset

The second dataset used in this dissertation is composed of HIV_{rt} (Human Immunodeficiency Virus reverse transcriptase) inhibitors selected from a recent review article [165]. The molecules were selected according to a single criterion; all molecules had EC50 values measured against MT-4 cells *in vitro*. The HIV_{rt} data set contains 64 molecules representing five structural classes of reverse transcriptase inhibitors. The five subsets are labeled as tibo(13) [166], hept(26) [167], tsao(11) [168], triazoline(7) [169], and thiadiazole(7) [170], the number in parentheses shows how many members from each respective class were included in the data set. To date QSAR models have been constructed for individual classes of HIV reverse transcriptase inhibitors typically using the comparative molecular field analysis (CoMFA) [171] or traditional QSAR methods [165]. However, a QSAR model encompassing several diverse classes has not appeared in the literature, with this in mind, the HIV_{rt} dataset was constructed to provide a serious challenge to our predictive modeling abilities.

The HIV_{rt} dataset is significant since it represents one of several ways that the spread of HIV and AIDS may be slowed or halted. These molecules inhibit an important biochemical step in the life cycle of the virus, and if inhibition were optimized, HIV would not be able to leave an infected cell and establish itself in another cell.

7.4 Caco-2 Dataset

The human intestinal cell line Caco-2 has been generally accepted as a primary absorption-screening tool in the early stage of drug development. In an effort to improve

lead generation hit-rates, there is great interest in defining quantitative relationships between molecular structure and the various modes of Caco-2 permeability. The Absorption, Distribution, Metabolism and Elimination/Excretion (ADME) characteristics of pharmaceuticals are important properties to be considered in the development of novel therapeutic agents. Many of the compounds entering into clinical trials often fail due to issues directly related to ADME. Among these properties, absorption is of paramount important in drug design. Currently, there is a heavy emphasis on producing drug candidates that have good oral absorption that hopefully extends to good oral bioavailability. Several *in vitro* test systems have been developed for measuring transport across intestinal mucosa. Of all *in vitro* systems that have been developed to date, the most commonly used method for determining the transport of compound across the intestinal mucosa currently involves the use of Caco-2 cells.

Caco-2 cell lines were originally established by Jorgen Fogh approximately 20 years ago after he screened several lines derived from human colon carcinomas [172, 173]. The Caco-2 cell line has been widely used for testing the GI absorption of compounds dues to its ability to express the morphological features of mature enterocytes. Therefore, they are amenable to rapid throughput screening for GI absorption. Measures of apparent permeation (Papp) are accepted as meaningful substitutes for actual human intestinal absorption values. Consequently, modeling of these permeability measures is an important substitute for modeling intestinal absorption.

In recent years, the number of predictive approaches to ADME using computational techniques has increased dramatically. Hybrid approaches, those combine computational techniques with easily obtainable experimental data, are also becoming more prevalent

[174]. The obstacle to good, general, and robust models of absorption has always been made to model absorption, and most are attempted with small datasets.

In this study, two representative Caco-2 cell permeability data sets were obtained from two literatures [175, 176]. These two structural heterogeneous data sets cover a relatively wide range of molecular size and lipophilicity.

Dataset I. Stenberg et al. reported Caco-2 cell permeability data for 27 structures [175]. All compounds are regarded as being transported by passive diffusion.

Dataset II. Another 48 structures are collected from Yazdanian et al. [176].

For each dataset, a large set of 780 descriptors was generated by combining electron-density based TAE, Property/Shape-Encode PEST, and the selected traditional MOE descriptors. Initial descriptor set contains 274 TAE descriptors [159], 396 PEST descriptors [160, 161], and 110 selected MOE descriptors that are related with pharmacophore, shape and volume.

CHAPTER 8

Correlation-based Feature Selection with Evolutionary Algorithms

Feature selection methods can be divided into two main categories based upon their feature subset evaluation: a *wrapper* or a *filter* approach [6]. A wrapper method searches for a good feature subset tailored to a particular predictive modeling approach and uses the induction algorithm as a black box for evaluating feature subsets. On the other hand, a filter method attempts to access the merits of features from the data alone. Wrapper methods generally use a learning algorithm with a statistical re-sampling method (such as cross validation, bootstrapping) to estimate the quality of feature subsets. This approach is useful but becomes computationally prohibitive because the learning algorithm must be trained and tested many times to evaluate feature subsets. On the other hand, filter methods are more practical than wrapper methods for applications to large datasets since they are much faster.

Evolutionary algorithms are heuristic search/optimization methods that provide robust and powerful adaptive search mechanism [177]. Evolutionary algorithms maintain a population of potential solutions that evolve according to rules of selection and some genetic operators such as recombination and mutation [177]. Individuals in the population are evaluated using a fitness function that mimics the environment. Individuals with high fitness value have a higher chance to survive and pass on their structures to the next generations.

Correlation-based feature selection with evolutionary algorithms will be explained in this chapter. This feature selection method can be thought as a filter method, which selects features based on the training data alone without taking the biases of learning algorithms into consideration [6]: i.e., the proposed evolutionary algorithms for feature selection are independent of the learning algorithm and are used as a filter to conduct a search for a good feature subset using a correlation-based evaluation function. The search space in a variable reduction problem with **T** features is $2^{\boldsymbol{T}}$ if all feature subsets are considered (including feature set with all features and no features). If the number of features to be selected is predefined, the optimal feature subset of size **N** chosen from a total of **T** features can in principle be found by enumerating and testing all possibilities based on a criterion, which requires $\binom{T}{N} = \frac{T!}{N!(T-N)!}$ tests. This becomes prohibitively expensive in computing time when **T** becomes larger. Evolutionary algorithms provide an alternative search method to select a good feature subset with a predefined size. Evolutionary algorithms explore the whole search space probabilistically by using a rank-based selection scheme and genetic operators (crossover and mutation) tailored to the representation of the individuals.

8.1 Correlation Based Evaluation Function

The evaluation function used by evolutionary algorithms, which will be explained in the next sections, is based on the hypothesis that *a relevant feature is highly correlated with the response variable(s) and less correlated with other features in the feature subset* [178]. Correlation is a bivariate measure of association (strength) of the relationship between two variables. In this dissertation, the correlation between variables **i** and **j** ($\mathbf{C_{ij}}$)

is measured in terms of the Pearson's correlation coefficient (r) that is a measure of the degree of linear association between two variables. It can take on the values from -1.0 to 1.0, where -1.0 is a perfect negative (inverse) correlation, 0.0 is no correlation, and 1.0 is a perfect positive correlation. The objective for feature selection can be related to maximizing the evaluation function defined in equation (8.1). The number of features to be selected, **N**, is predetermined. The fitness function for the individual **k** containing **N** features is defined by

$$F_k = \left(\frac{1}{N}\right)\sum_{i=1}^{N} |C_{iR}| - \alpha\left(\frac{2}{N(N-1)}\right)\sum_{i=1}^{N}\sum_{j=i+1}^{N} |C_{ij}| - \beta \qquad (8.1)$$

where:

$\mathbf{F_k}$ = Fitness of individual **k**, k = 1, 2, 3, …, Pop_size.

$|\mathbf{C_{ij}}|$ = Absolute value of the inter-correlation between feature **i** and feature **j**.

$|\mathbf{C_{iR}}|$ = Absolute value of the correlation between feature **i** and the response variable **R**.

a = Inter-correlation penalty factor.

b = Death penalty factor: i.e., If $|C_{ij}| > 0.95$ then $\beta = 1000$; otherwise $\beta = 0$.

In the fitness function, the sums of correlations are scaled in terms of the number of features in order to prevent trivial solutions. a is a user defined penalty factor for inter-correlation and takes on values between zero and one. It is obvious that when a = 0, the objective function is simply to find the features which are the most highly correlated with the response variable. If a is greater than zero the inter-correlated variables are penalized. b is a parameter called death penalty factor, which gives a very high penalty to an

individual having features with inter-correlation higher than 0.95. The death penalty factor allows us to define an upper threshold for inter-correlation: because these features are usually highly correlated, we use the term “cousin features” in this context. Those feature subsets with features inter-correlated above this threshold are penalized. This parameter also assures that individuals with duplicated features die off in next generations.

8.2 Rank-Based Selection Scheme

Evolutionary algorithms for feature selection in this thesis use a rank-based selection scheme. Rank selection, a nonparametric procedure for selection, was introduced by Baker [52]. In this method individuals in the current population are sorted according to their fitness values and individuals for the next generation are selected proportionally to their rank rather than their actual objective function values. Ranking acts as a function transformation that assigns a new fitness value to an individual based on its performance relative to other individuals [53]. This method prevents the super individuals to take over population in a few generations by adjusting the ranking weights.

8.3 Genetic Algorithms for Feature Selection (GAFEAT)

A key issue in applying GAs to any problem is the selection of an appropriate formulation and representation that represents all possible solutions to the problem. In a feature selection problem the objective is to represent the space of all possible subsets of the given feature set. The traditional and simplest representation is a binary representation, where each chromosome consists of fixed-length binary string with a size

the number of features in the problem. Each bit in the chromosome represents either the elimination or the inclusion of the corresponding feature.

Binary string representation has some drawbacks. In a typical binary GA for feature selection, one of the important issues is the initialization of the initial population. Since a chromosome consists of zeros and ones (representing exclusion and inclusion of the corresponding feature), deciding number of "1" in individuals of the initial population is problematic. In binary GAs for feature selection, the value of each gene in a chromosome is determined probabilistically by the toss of a coin. If the coin were fair (no bias) an average of 50 percent of the genes in a chromosome would have a value of "1", which means that corresponding features are included. The number of "1" is controlled by introducing biases into selection of gene values. Leardi, et al. [7] suggest a biased probability producing low number of "1" in chromosomes of the initial population. They suggest that a good value for this probability is a value that selects an average of five variables per chromosome (e.g. 10 percent when working with a data set with 50 variables). The GA itself builds more complex combinations with more features through crossover and mutation operators.

Although binary representation is able to represent all possible feature subsets, it can still cause premature convergence. Consider a data set with 10 features. The whole search space for this problem is $2^{10} = 1024$ if all feature subsets are considered. If the number of features to be selected is predefined with a size N (where N=1, 2, 3, …, 10), the search subspace will be

$$\binom{T}{N} = \frac{T!}{N!(T-N)!}.$$

The sum of all subspaces will be equal to whole search space.

$$\sum_{N=0}^{T} \frac{T!}{N!\,(T-N)!} = 2^{T},$$

where T is the total number of features. Table 8.1 shows search subspaces and their relative percentages for the dataset with 10 features. Figure 8.1 shows the relative size of the search subspaces to total search space. It is now obvious why a binary GA may convergence to a suboptimal solution. For instance, for subspaces with 2 features and 5 features the total search space is 45 and 252, respectively. There is a high chance for the GA to converge to a solution that is optimal (or close to optimal) solution for the subspace with 2 features, even though the global optimal solution may reside in the subspace with 5 features. Leardi, et al. [7] suggest a stratified selection scheme in which only chromosomes belonging to the same subspace compete with each other for the next generation. Although this is a good strategy to overcome the premature convergence, there is still some chance that some of the subspaces may not be adequately represented in the population.

Even with the stratified selection scheme proposed by [7], the classical binary crossover and mutation operators may produce offspring, which belong to a different subspace than that of the parent chromosomes. This causes a loss of information gained by the GA for the current subspaces. Thus, binary genetic operators may cause to poor convergence since the GA cannot exploit the information properly for the stratified subspace. Binary operators are illustrated in the Figure 8.2.

Table 8.1 Search Subspace for a Dataset with 10 Features

Number of features to be selected (N)	Search space	Percentage search space
0	1	0.0010
1	10	0.0098
2	45	0.0439
3	120	0.1172
4	210	0.2051
5	252	0.2461
6	210	0.2051
7	120	0.1172
8	45	0.0439
9	10	0.0098
10	1	0.0010
Total	1024	1.0000

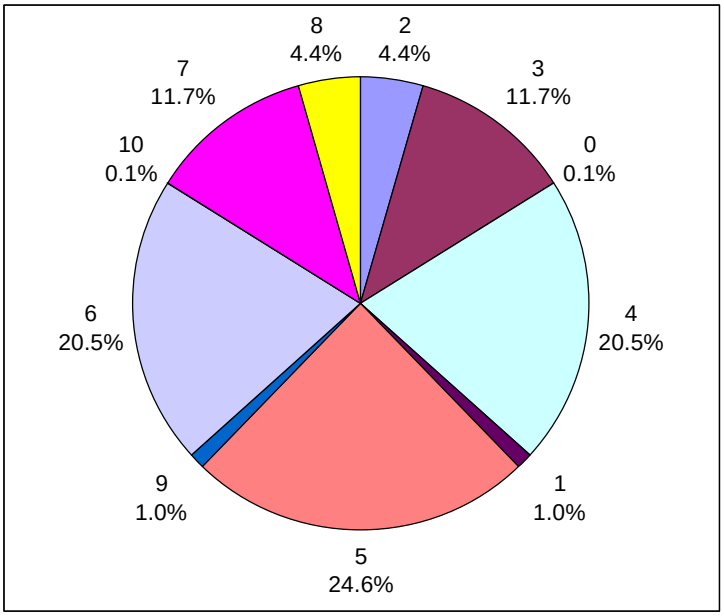


Figure 8.1 Percentage Search Subspace for a Dataset with 10 Features

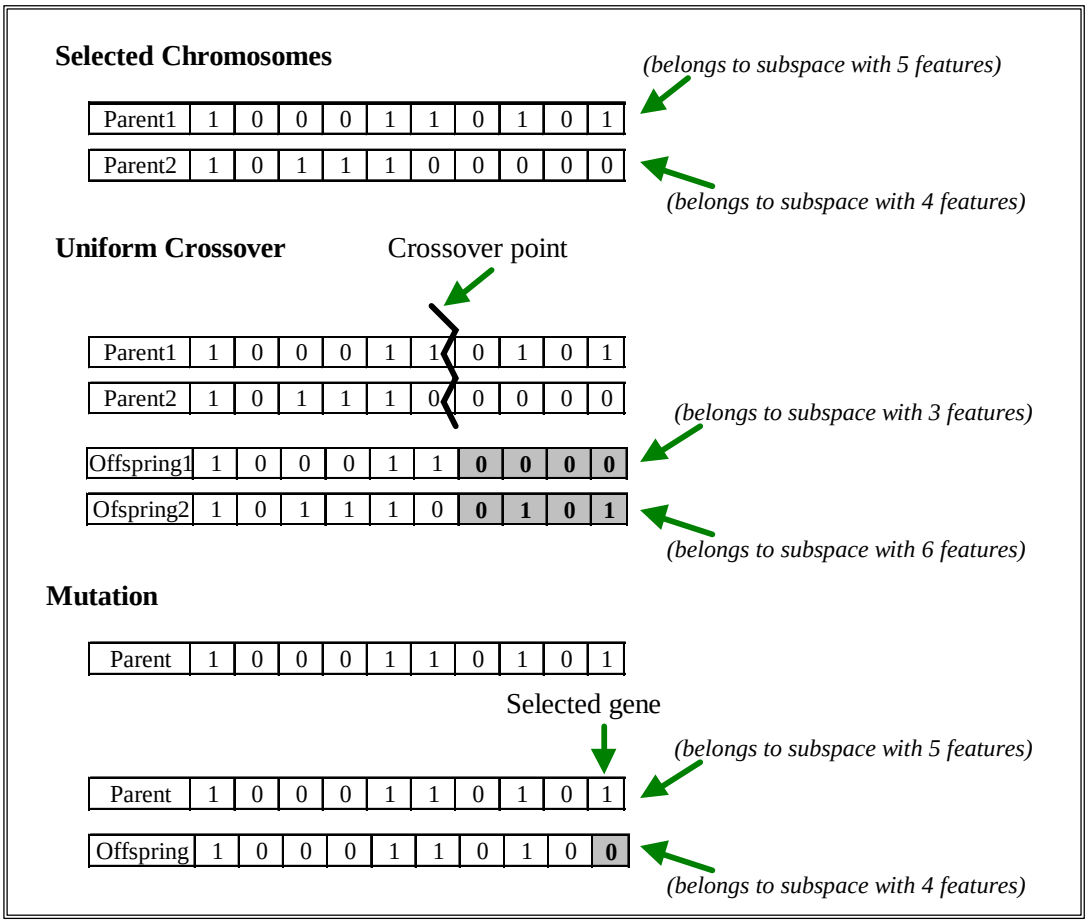


Figure 8.2 Binary Crossover and Mutation Operations for Feature Selection Problem

Because the classical binary GAs are not well suitable for the feature selection problem, GAs based on floating-point and unique list representations are proposed. These representations and their genetic operators are explained in the following sections.

8.3.1 Floating-Point Representation

A genetic algorithm (GA) for feature selection with a floating-point representation for the features is proposed. The number of features (N) to be selected is pre-specified. Chromosomes are represented as floating point arrays with size N in which each gene

corresponds to the variable number in the feature subset. The initial population is created randomly as floating point arrays where each entry (gene) in an array (chromosome) is created using the following function:

$$Gene_{ij} = (U_{ij} * T) + 1,$$

where $\mathbf{U_{ij}}$ is an uniform random number between zero and one, **T** is the total number of features in the dataset under study. The index, **ij** represents the $\mathbf{j}^{th}$ gene position on the $\mathbf{i}^{th}$ chromosome in a population. For example for the case where 10 features are to be selected out of total of 100 features, an individual (genotype) could be represented as

93.57, 64.31, 6.96, 54.40, 38.97, 60.43, 13.25, 48.91, 30.97, 4.03

in which the integer part of each floating-point number (gene) represents the corresponding feature. The phenotype of the chromosome represented by the above floating point array will be:

93	64	6	54	38	60	13	48	30	4

In this genotype-phenotype mapping, different genotypes can correspond to the same phenotype. Also, the genotype-phenotype mapping can lead to illegal feature subsets (individuals), which means that some individuals may have some duplicated features. The proposed GA corrects illegal individuals after creating the random initial population. Hence, initial population consists of legal individuals only.

8.3.2 Crossover and Mutation for Floating-Point Representation

The proposed floating-point GA uses classical one-point crossover as originally proposed by Holland [32]. This crossover operator simply chooses a random position on two chromosomes (strings), divides strings into two parts, and swaps the tails of the strings between them. For this study, the genes of the genotype of each individual are sorted in ascending order before the crossover operation. Such a sorting reduces the chance of having illegal individual after crossover operation. The GA does not try to correct the illegal individuals, but penalizes those individuals to make sure that they will die off in the next generation. The penalty method is the most common method to handle infeasible (illegal) solutions in the evolutionary algorithms for simple constrained optimization problem. The addition of a penalty term to the objective function transforms the constrained optimization problem into an unconstrained optimization problem.

Crossover is an expensive operator for the feature selection problem since some individuals may violate some of the constraints of the problem. Figure 8.3 illustrates the crossover operation for a problem where in which the objective is to select a good subset with 10 features out of total of 100 features. The mutation operator chooses a random gene position and changes the value of gene within the feature range. Note that the mutation operator may also lead to an illegal individual.

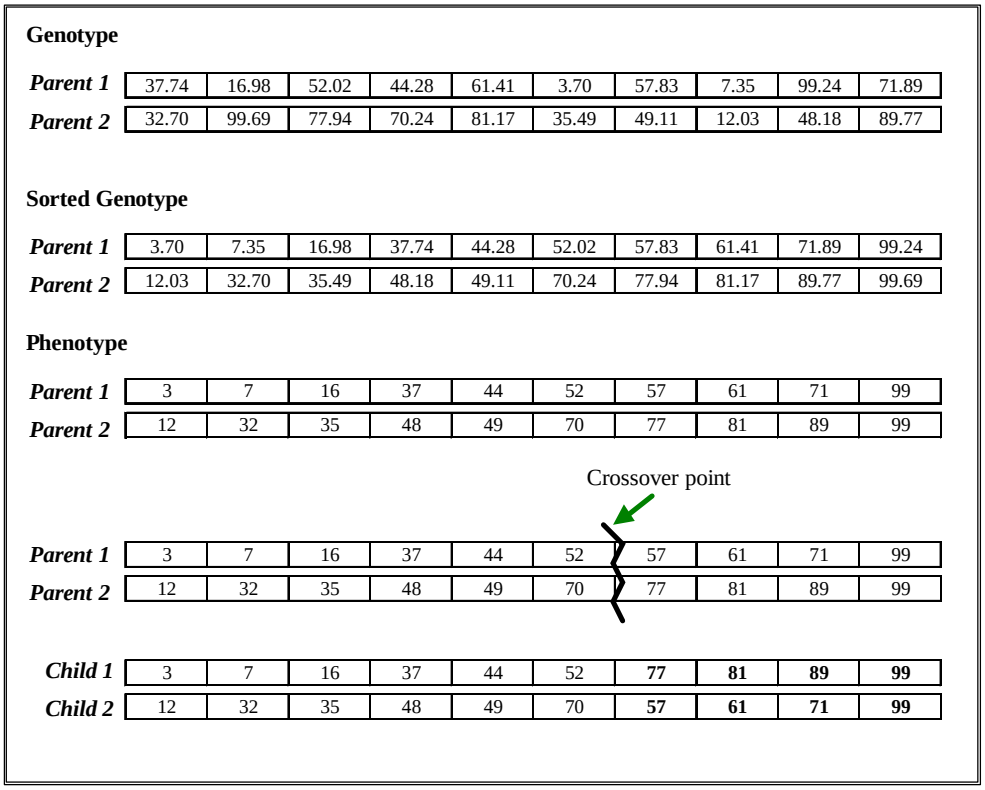


Figure 8.3 One-point Crossover Operator in the Floating-point GA for Feature Selection

8.3.3 Unique List Representation

Another alternative for the phenotype representation employed by GAFEAT is a unique list representation. Given a dataset containing T features, each chromosome represents a legal subset containing N features. In this representation, a chromosome is as an integer array of size N. GAFEAT always works on the phenotype individuals after creating an initial population. With no duplicated features a unique list representation can be thought of as an order-based chromosome where N features represented in a chromosome are the selected features, and the remaining (T-N) features not in the chromosome are the non-significant features.

8.3.4 Crossover and Mutation operators for Unique List Representation

Partially Mapped Crossover (PMX) was introduced in Chapter 3. PMX, proposed for the Traveling Salesman Problem by Goldberg and Lingle [1], produces an offspring by choosing a portion of tour from one parent and preserving the position and relative order of as many cities as from the other parent [9]. PMX can be viewed as an extension of two-point crossover with a special repairing mechanism to resolve the illegitimacy caused by two-point crossover. PMX can be adapted to the feature selection problem. The procedure for the PMX is the following:

Step 1. Select parents for crossover.

Step 2. Select randomly two crossover points on the chromosomes. The sections between two crossover points are called the mapping sections.

Step 3. Swap the mapping sections between parent chromosomes.

Step 4. Determine the mapping relationship between mapping sections.

Step 5. Build offspring based on mapping relationship.

This procedure is illustrated in Figure 8.4.

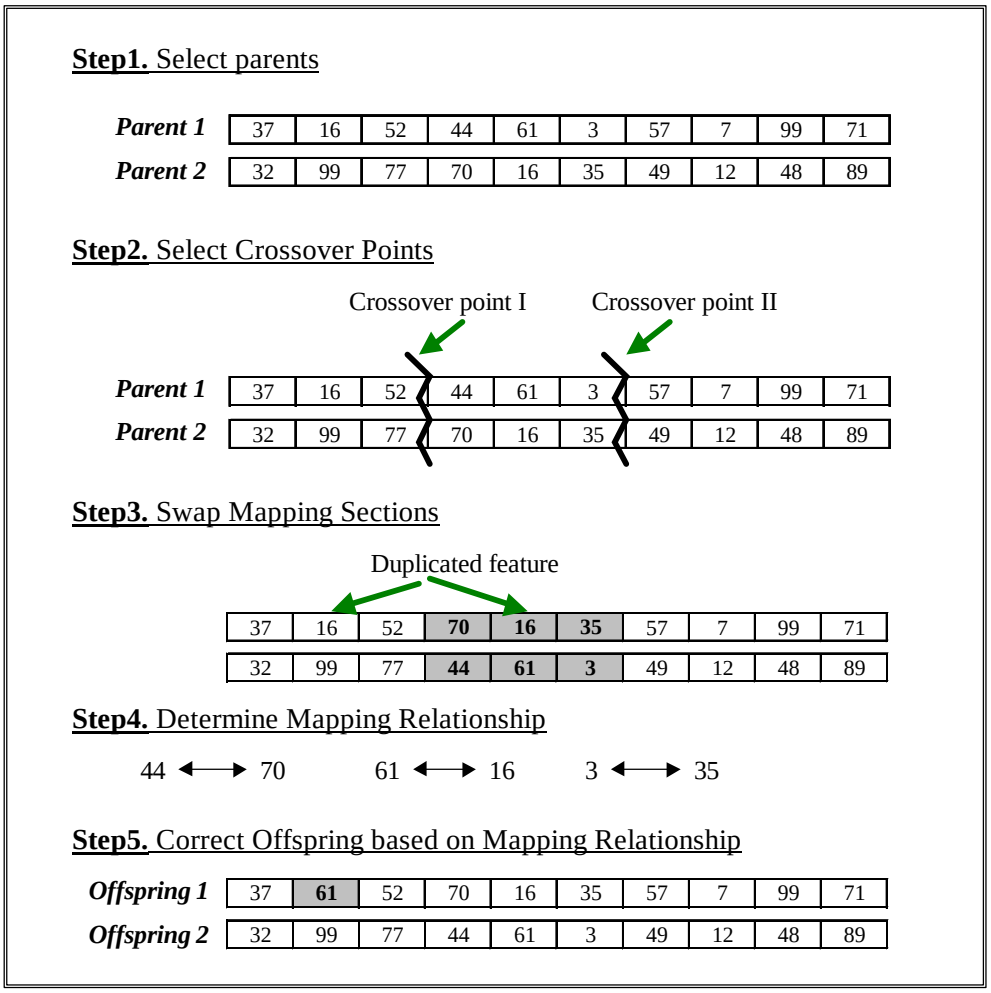


Figure 8.4 Partially Mapped Crossover for the GA with the Unique List Representation for Feature Selection

In order to avoid illegal offspring from mutation, the mutation operator now randomly chooses a gene position in the chromosome and tries to exchange it with a unique random value. If a unique value cannot be found after a certain number of trials then the parent chromosome is copied without any change.

8.4 Evolutionary Programming for Feature Selection (EPFEAT)

Evolutionary Programming (EP) was explained in section 2.4.2 and section 4.6 and successful implemented for the Traveling Salesman Problem in section 4.6. EP traditionally used representations that are tailored to the problem domain [177]. The most important difference between a GA and EP is that EP does not use any recombination (crossover) operator. The forms of mutation used in EP are quite flexible and can produce perturbations similar to recombination if intelligent mutation operators are devised. Therefore, the performance of an EP is affected by its choice of the mutation operators used to create variability and novelty in evolving populations [177].

8.4.1 Representation for EPFEAT

An evolutionary programming algorithm for feature selection problem (EPFEAT) is proposed based on the unique list representation introduced in section 8.3.3. Given a dataset consisting of total **T** number of features, each chromosome corresponds to one of the permutations of an array with the numbers 1 - **T**. This means that each chromosome has a different ordered list of **T** features similar to the Traveling Salesman representation used in Chapter 4. Since the number of features to be selected (**N**) is predetermined, the first **N** features of a chromosome are used for calculating the fitness of the chromosome. This can be thought of as an order-based list, where the left-most **N** features are the selected features and, the remaining **T**-**N** values correspond to unselected features. The representation of an individual is illustrated in Figure 8.5. In Figure 8.5, the objective of the EP algorithm is to select 4 features out of 10. An individual represents the dataset

with a total of 10 features, where the first 4 features construct the feature subset and the remaining features (the shaded features) are the non-significant feature subset.

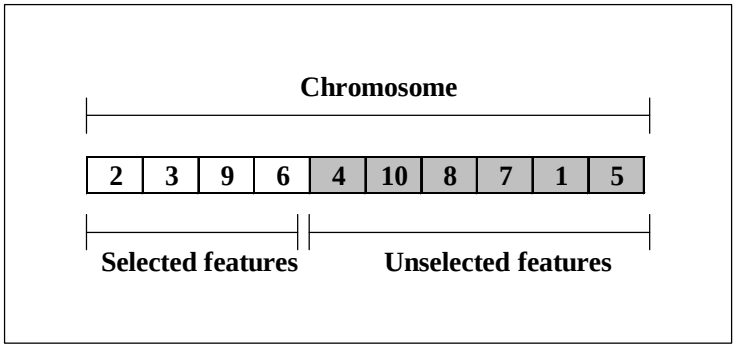


Figure 8.5 Representation for Evolutionary Programming Algorithm for Feature Selection

8.4.2 Mutation Operator for EPFEAT

A new mutation operator for EPFEAT is proposed. The first N features are used for measuring the fitness of the individuals. This can be considered as dividing the total number of features into two subsets based on a fitness function. These two feature subsets can be thought of further as sub-chromosomes. The proposed mutation operator randomly selects a gene position within the first N features and within last (T-N) features and swaps these features. This operator can be considered as a recombination operation between two sub-chromosomes. One of the most important consequences of this representation is that it allows us to develop a mutation operator that produces perturbation similar to crossover operator. Note that this mutation operator always produces a legal feature subset (i.e., no duplicate features).

8.5 Comparisons of Evolutionary Algorithms for Feature Selection

The proposed three evolutionary algorithms for feature selection (genetic algorithm with the floating-point, genetic algorithm with the unique list representation, and the evolutionary programming algorithm) are compared using the HIV_{rt} dataset with 64 molecules and 620 features explained in section 7.3. For comparison, the parameter a and the threshold correlation for the parameter b in the fitness function are set to 0 and 1.0, respectively, leading the fitness function in equation (8.1) for the most correlated features only according to:

$$F_k = \left(\frac{1}{N}\right)\sum_{i=1}^{N} | C_{iR} |$$

In other words, the objective of the feature selection algorithms is simply to find N features that are highly correlated with response variable. The optimal solution for this problem is easy to find by calculating correlations between features and response variable and sorting them in descending order. The performances of the proposed algorithms are then compared with the optimal solution. Each algorithm was executed 10 times to select 25 out of 620 features with a different initial population (i.e., a different random seed). The parameter settings of the algorithms are shown in Table 8.2.

Table 8.2 Parameter Settings for the Evolutionary Algorithms for Feature Selection

Algorithm	Population	Crossover Probability	Mutation Probability
GAFEAT (Floating-point)	200	0.90	0.40
GAFEAT (Unique List)	200	0.90	0.02
EPFEAT	200	None	0.02

Each algorithm was able to find the 25 features most correlated with the response variable in each run. The results of those computational experiments are presented in Table 8.3. These results show that the proposed representations are very efficient.

Table 8.3 Comparisons of Genetic Algorithm with Floating-point and Unique List Representations, and Evolutionary Programming Algorithm for Feature Selection

25 features most correlated with response			GAFEAT-Floationg-point Representation										GAFEAT-Unique List Representation										EPFEAT									
No	Feature No	Feature Label	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	241	Del.G.NIA	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2	331	PIPAvg	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
3	267	SIGIA	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
4	191	Del.K.IA	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
5	402	LaplAvg	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
6	150	pips6	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
7	217	SIKIA	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
8	166	Del.Rho.NIA	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
9	158	pipd6	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
10	347	PIP16	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
11	393	AbsFuk5	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
12	507	SHsOH.91	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
13	544	CHsOH.182	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
14	570	CsOH.227	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
15	149	pips5	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
16	533	SsOH.136	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
17	260	AbsDGN7	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
18	390	AbsFuk2	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
19	575	CdS.242	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
20	538	SdS.151	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
21	348	PIP17	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
22	391	AbsFuk3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
23	332	PIP1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
24	159	pipd7	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
25	487	dxvp4.71	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Run Time (in second)			62	63	63	60	61	60	61	62	61	61	99	98	98	100	98	97	98	98	97	98	132	133	132	132	132	132	133	134	133	132

8.6 Effects of the Inter-correlation Penalty (a) Factor on Variable Selection

In this section, the effects of the inter-correlation factor on variable selection will be investigated. For this purpose, a synthetic (i.e., artificially made-up) dataset with 64 data points and 600 descriptive features was randomly drawn from a normal distribution with mean 0 and standard deviation 1. Values of the response variable were constructed as a linear combination of first 40 features from this data set. The coefficients of these 40 variables were chosen randomly. Since the 40 features are known in advance, the

performance of GAFEAT can be tested by selecting 40 features from the dataset with different a values (ranging between 0 and 1). Multiple Linear Regression is used in order to assess the quality of the feature subsets with 40 features selected by GAFEAT. The qualities of the feature subsets are reported based on the Multiple Coefficient of Determination (R^2), the F statistic and the P value of F statistic. These statistics allow us to assess or judge the usefulness of the regression model (feature subset). R^2 is a statistic widely used to determine how well a regression model fits to data. The F statistic is the test statistic used to decide whether the model as a whole has statistically significant predictive capability. A large value of F (or a low value of P) as well as a large value of R^2 indicates that most of the variation of the response variable is explained by the regression equation and that model is useful [179]. The results of these experiments are presented in Table 8.4. It is apparent from Table 8.4 that when α increases the quality of the features subset increases, even though the number of original variables included in the feature subsets decreases. The R^2 of the regression models constructed with a subset of 40 features selected by GAFEAT for this synthetic dataset is plotted versus α values in Figure 8.6.

Table 8.4 Results for 40 Features Selected by GAFEAT from Synthetic Dataset at Different α Values

a	#of Selected Original Variable	R^2	F value	P value
0.00	7	0.9143	7.3487	0.0000
0.10	8	0.9331	9.2480	0.0000
0.20	7	0.9350	9.3277	0.0000
0.30	7	0.9344	9.3496	0.0000
0.40	6	0.9361	8.9800	0.0000
0.50	6	0.9390	9.3949	0.0000
0.60	6	0.9427	11.2437	0.0000
0.70	6	0.9434	11.1698	0.0000
0.80	6	0.9432	11.1622	0.0000
0.90	6	0.9439	11.1552	0.0000
1.00	6	0.9533	13.7375	0.0000

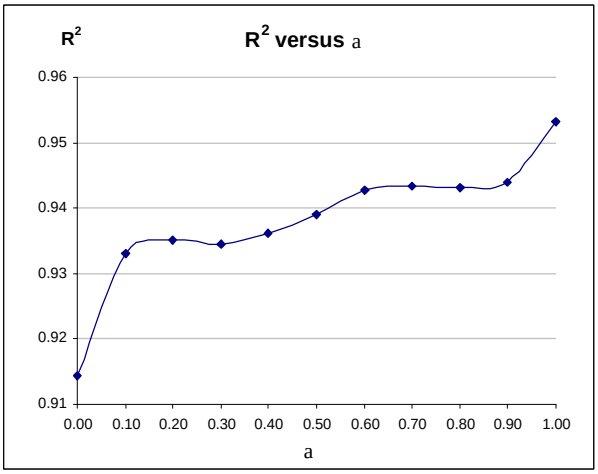


Figure 8.6 The Multiple Coefficient of Determination (R^2) of the Feature Subset with 40 Features Selected by GAFEAT at Different α Values from Artificial Dataset versus Inter-correlation Factor (α).

The coefficient (weight factor) of each of the original 40 features used to construct to the response variable is shown in the Table 8.5. The shaded six features in the Table 8.5 were always selected by GAFEAT at all of the different alpha values. The common properties of these six features (features 16, 29, 17, 28, 39, and 19) are that they have higher coefficients and are highly correlated with the response variable. Selecting the 40 features that are the most correlated with response variable (i.e., $\alpha = 0$) contains 7 of the original 40 features; its performance in R^2 is lower than selected feature subsets with α values exceeding 0.4. As can be seen from Figure 8.6, this indicates that penalizing the inter-correlation between features while maximizing their correlation with the response variable helps to construct a better regression model.

Table 8.5 Weight Factors, Correlation Ranks and Correlations with the Response Variable of 40 Features that Constructed to the Response Variable for Random Coefficients

Feature No	Coefficient	Correlation with Response Variable	Correlation Rank
16	-0.5469	-0.3723	4
8	-0.4414	-0.0895	288
29	0.3675	0.5188	1
27	0.3439	0.0764	329
17	0.2775	0.4085	2
4	0.2759	0.0127	551
28	-0.2737	-0.2349	41
14	0.2100	-0.1162	222
35	-0.2061	-0.1555	138
1	0.2034	-0.0241	507
15	0.1913	0.1343	186
39	0.1768	0.2811	15
19	0.1765	0.2421	36
9	0.1709	0.0813	317
25	0.1636	0.1385	178
33	0.1496	-0.0942	279
26	-0.1482	-0.1013	262
38	0.1244	-0.1198	215
7	0.1226	-0.0586	387
3	0.1117	0.1587	133
20	-0.1070	-0.1067	243
36	0.1007	0.0820	311
6	0.0978	-0.2355	40
23	0.0836	0.0165	535
30	-0.0766	-0.1544	141
18	-0.0641	-0.0913	285
40	-0.0636	0.0760	331
2	0.0624	0.0705	350
34	-0.0618	-0.2522	31
24	-0.0527	-0.1535	142
10	0.0516	-0.0031	590
21	-0.0493	-0.0058	581
11	-0.0416	-0.0726	341
5	0.0390	-0.0727	340
22	0.0249	-0.0685	356
31	0.0106	-0.0560	396
32	0.0099	0.1109	233
13	0.0064	-0.1052	250
37	-0.0005	0.0712	345
12	0.0000	-0.0413	447

Figure 8.7 depicts the different a values versus the average correlation between the selected features (FF) and the average correlation between response variable and the selected features (RF). When the inter-correlation value (a) increases FF and RF decrease but the amount of decrease in FF exceeds that of RF. This is not easily seen from Figure 8.7 since the dataset was randomly created. Therefore, the difference between RF and FF versus α is plotted in Figure 8.8.

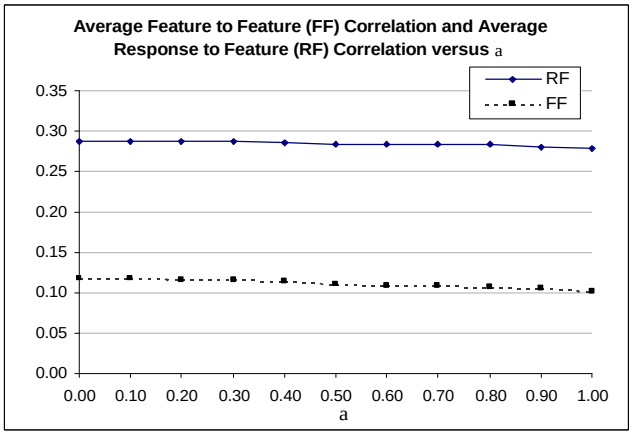


Figure 8.7 Inter-correlation Penalty (α) versus Average Feature to Feature (FF) Correlation and Average Response to Feature (RF) Correlation for Artificial Dataset

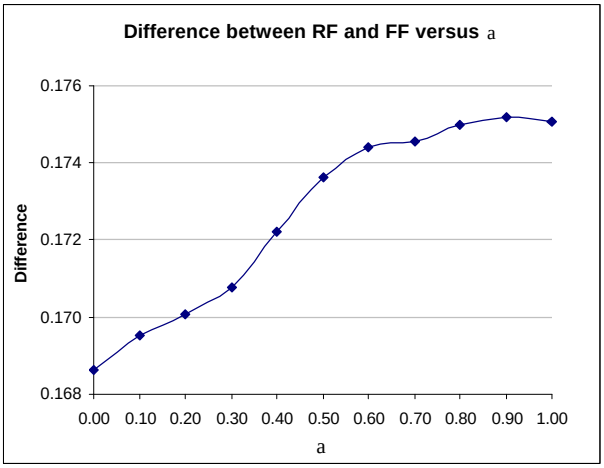


Figure 8.8 Inter-correlation Penalty (α) versus Difference between the Average Response to Feature (RF) Correlation and the Average Feature to Feature (FF) Correlation and for Artificial Dataset

The effect of the inter-correlation factor (α) on variable selection was also investigated by keeping the coefficients of all 40 variables at the same value in the construction of the values of the response variable. The results of these experiments are shown in the Table 8.6. The number of original features selected by GAFEAT is 10 for α values of 0.0, 0.10, 0.20, 0.30, 0.40, 0.50, and 0.60. Although the R^2 values of regression models fluctuate, there is an increasing trend in the R^2 values when α increases. On the other hand, the number of original features selected by GAFEAT is 11 at α values of 0.80, 0.90, and 1.00 and R^2 values tend to increase monotonically with α. Figure 8.9 depicts R^2 versus a for the case where feature coefficients are set to same value.

Table 8.6 Results for 40 Features Selected by GAFEAT from Artificial Dataset (constant weight) at Different Alpha Values

a	#of Selected Original Features	R^2	F value	P value
0.00	10	0.915	6.1528	0.0000
0.10	10	0.930	7.6307	0.0000
0.20	10	0.928	7.3758	0.0000
0.30	10	0.921	5.7778	0.0000
0.40	10	0.930	6.0760	0.0000
0.50	10	0.930	7.5784	0.0000
0.60	10	0.936	6.1712	0.0000
0.70	10	0.931	6.4777	0.0000
0.80	11	0.937	8.5247	0.0000
0.90	11	0.957	12.8164	0.0000
1.00	11	0.961	8.5493	0.0000

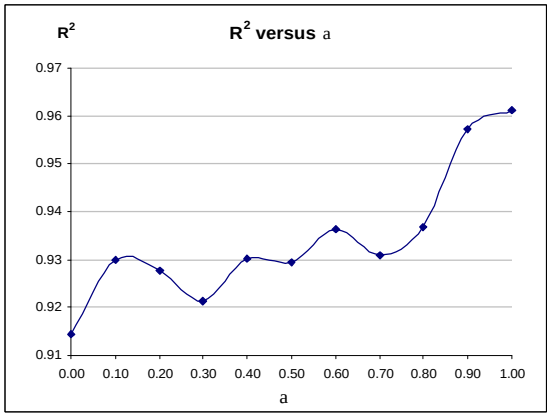


Figure 8.9 Multiple Coefficient of Determination (R^2) versus Inter-correlation Factor (α) for Same Coefficients

Table 8.7 Weight Factors, Correlation Ranks and Correlations with the Response Variable of 40 Features that Constructed to the Response Variable with the Same Coefficients

Feature No	Coefficient	Correlation with Response Variable	Correlation Rank
26	0.5	0.3717	2
37	0.5	0.3363	3
28	0.5	0.3271	6
40	0.5	0.3159	7
1	0.5	0.3140	8
16	0.5	0.2759	17
34	0.5	0.2616	21
6	0.5	0.2601	22
24	0.5	0.2557	26
25	0.5	0.2496	27
15	0.5	0.2042	70
23	0.5	0.1974	82
10	0.5	0.1962	86
14	0.5	0.1934	87
11	0.5	0.1933	88
30	0.5	0.1869	95
31	0.5	0.1841	99
3	0.5	0.1782	106
21	0.5	0.1742	110
2	0.5	0.1709	118
39	0.5	-0.1396	167
5	0.5	0.1387	169
22	0.5	0.1377	171
20	0.5	0.1363	174
9	0.5	0.1203	208
7	0.5	0.1189	210
33	0.5	0.1110	229
17	0.5	0.1101	232
36	0.5	0.1097	233
12	0.5	0.1046	249
18	0.5	0.1040	252
27	0.5	0.0866	286
32	0.5	0.0710	344
35	0.5	0.0582	386
4	0.5	0.0520	409
19	0.5	0.0287	494
8	0.5	-0.0230	510
29	0.5	-0.0229	511
38	0.5	-0.0055	566
13	0.5	-0.0001	600

Weight factors, correlation ranks and correlations with the response variable of 40 features that were used to construct the response variable are presented in Table 8.7. The gray shaded nine features (26, 37, 28, 40, 1, 16, 34, 24, and 25) are common to the feature subsets selected by GAFEAT for all α values. These features are highly correlated with the response variable, which are amongst the top 30 most correlated features to the response variable.

8.7 Effect of the Inter-correlation Penalty (α) Factor on Prediction

In this section, the effect of the inter-correlation penalty factor (α) on the predictive performance of learning algorithms (Multi-Layered Perceptrons and Partial Least Squares) will be investigated. For this purpose, the HIV_{rt} dataset (see section 7.3) with the set of 160 wavelet descriptors [163] explained in section 7.1.1 was employed.

The neural network model used in all methods is a standard feed-forward multi-layered perceptrons trained with the back-propagation algorithm [140, 141]. The artificial neural networks have two hidden layers and are oversized (the hidden layers for this study contained 13 and 11 neurons respectively). Training was halted with early stopping by policy (when LMS error drops below at 0.09). Because of the early stopping procedure, the neural network results are not very sensitive to the number of neurons in the hidden layers. On the other hand, because of the early stopping the neural network models will also be relatively linear. The PLS models described in chapters of this dissertation used four latent variables for all calculations.

The predictive performance of the learning algorithm heavily depends on the appropriate value of α. Since the performances of the evolutionary algorithms proposed

in this Chapter are more or less similar, GAFEAT with a floating-point representation will be used for feature selection. Results of feature selection with GAFEAT will be compared with those from the Sensitivity Analysis explained earlier in section 6.5.

8.7.1 Evaluation of Learning Algorithms

The learning algorithms for model building are Multi-Layered Perceptrons (MLPs), trained with back-propagation and Partial Least Square (PLS) regression. The MLP and PLS algorithms are both implemented in the StripMiner^{TM} [4]. Estimating the error of a learning model constructed from a set of training data is important to predict its performance on future unseen data and/or to compare it with its competitors. Cross-validation and bootstrapping are methods for estimating the generalization error rate of a learning model based on resampling [157]. In a cross-validation procedure, the dataset is randomly divided into ***k*** non-overlapping subsets where the size of each subset is as equal as possible. The learning model is trained and tested ***k*** times by using the ***j*** th (where ***j***=1, 2, …, ***k***) subset as a test set (i.e., validation set, both the terms, test set and validation set, will be used interchangeably for the out of sample) and combining the remaining subsets together as a training set.

Bootstrapping, introduced by Efron [156], is a general technique for estimating sampling distributions. In bootstrapping, multiple samples (anywhere from 50 to 2000 samples) of data of size ***n*** are taken uniformly from the original data of size ***n*** with replacement. Since bootstrap samples are drawn with replacement, the probability of any given instance not being part of the bootstrap dataset is $(1-1/n)^n = 1/e = 0.368$, which can be thought of as smoothed versions of cross-validation [157]. In our implementation, the

(***n-v***) samples for training set are drawn without replacements, where ***n*** is the number of samples in the dataset and ***v*** is the number of samples in the validation set. There is no overlap between training and validation sets for each bootstrap sample. The model is trained on the training set and the error rate for the learning model is estimated on the error on the validation set. The performance estimations of different learning algorithms are reported with two distinct measures for the error: $\mathbf{q^2}$ and $\mathbf{Q^2}$ based on the validation set for each bootstrap sample.

8.7.2 Effect of the a on the Prediction Quality

GAFEAT was run with different a values ranging between 0 and 1 to select 40 descriptors from the HIV_{rt} dataset with 160 wavelet descriptors. Each selected set of descriptors is used to construct a predictive model using MLPs and PLS regression. Error rates of the learning algorithms are calculated based on 100 bootstrap samples in terms of q^2 and Q^2 by leaving 58 molecules in the training set and 6 molecules in validation set. Figures 8.10 and 8.11 show how the a value affects the predictive performance of MLPs and PLS regression, respectively. One conclusion that can be easily drawn from these results is that the selection of the inter-correlation factor is a critical issue for choosing a good subset of descriptors. The low values as well as high values for inter-correlation penalty factor deteriorate the prediction error of the MLPs and PLS regression. For the HIV_{rt} dataset, 0.55 for was found to be optimal for α. Table 8.8 presents the results of the 100 bootstraps validation errors of the MLPs and PLS regression with all 160 features. According to these results, if the right α value is applied, GAFEAT can select relevant

features that lead to better predictive models (compared to models that employ all the available features).

Table 8.8 100-bootstrap Validation of the HIV_{rt} Dataset with 160 Wavelet Descriptors

Learning Models	Validation Error	
	q^2	Q^2
PLS	0.4294	0.4377
MLP	0.4190	0.4400

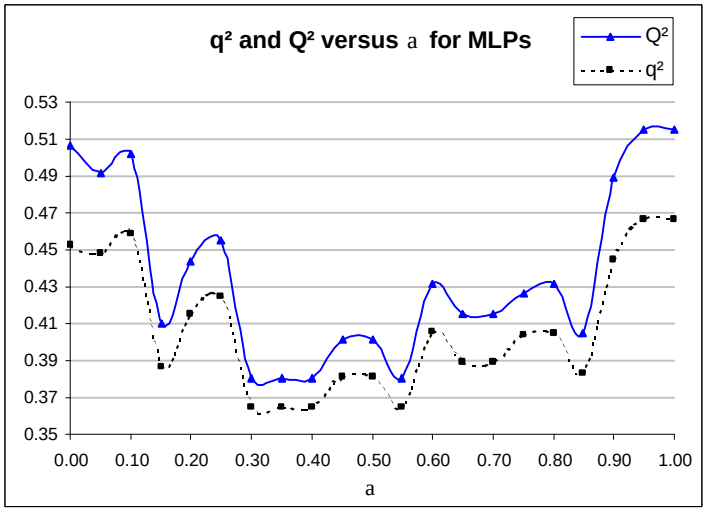


Figure 8.10 Inter-correlation Penalty Factor versus q^2 and Q^2 of MLPs Models Constructed by 40 Features Selected by GAFEAT from HIV_{rt} Dataset with 160 Wavelet Descriptors. The values of q^2 and Q^2 are based on 100-bootstrap Samples

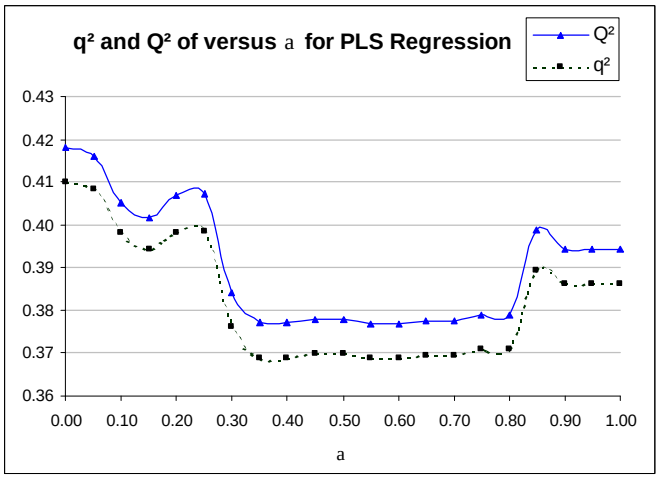


Figure 8.11 Inter-correlation Penalty Factor versus q^2 and Q^2 of PLS Models Constructed by 40 Features Selected by GAFEAT from HIV_{rt} Dataset with 160 Wavelet descriptors. The values of q^2 and Q^2 are based on 100-bootstrap samples

Figure 8.12 depicts the average correlation between the selected features (FF) and the average correlation between response variable and the selected features (RF) versus α. When α increases FF and RF decrease, but the amount of decrease in FF is higher than that of RF. It can be seen from Figure 8.12, that when α is around 0.55, feature subsets are less correlated with each other and more correlated with the response variable.

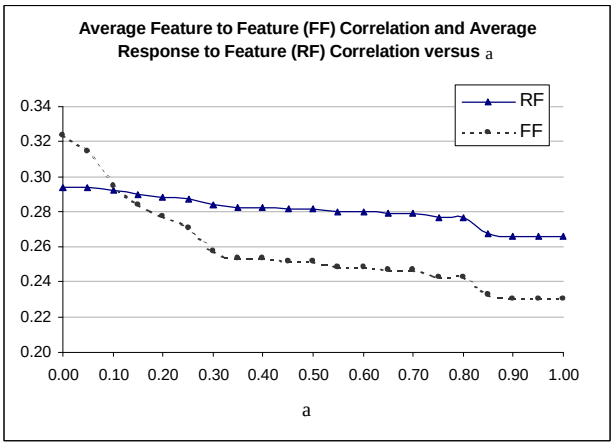


Figure 8.12 Inter-correlation penalty (α) versus Average Feature to Feature (FF) Correlation and Average Response to Feature (RF) Correlation for HIV_{rt} Dataset with 160 Wavelet Descriptors

The correlation matrix of the data with 40 selected features (ordered by correlation with the response) at α value 0.55 is presented in Figure 8.13. Here, the last column and row correspond to the response variable. Darker colors represent a high correlation value between the two corresponding variables. It can be noticed that a few features are highly inter-correlated and that the selected features are better correlated with the response variable. Figures 8.14-A and 8.14-B show the histograms of the correlation of the response variable with all 160 descriptors and with 40 selected descriptors by GAFEAT, respectively.

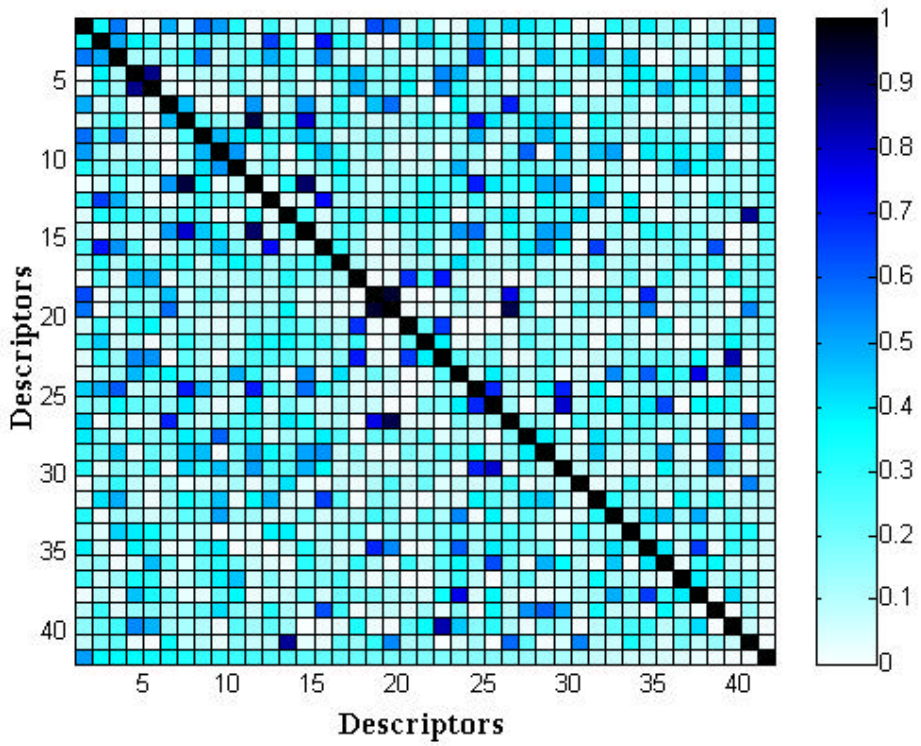


Figure 8.13 Colorplot of Correlation Matrix of 40 Features Selected by GAFEAT from HIV_{rt} dataset with 160 Wavelet Descriptors

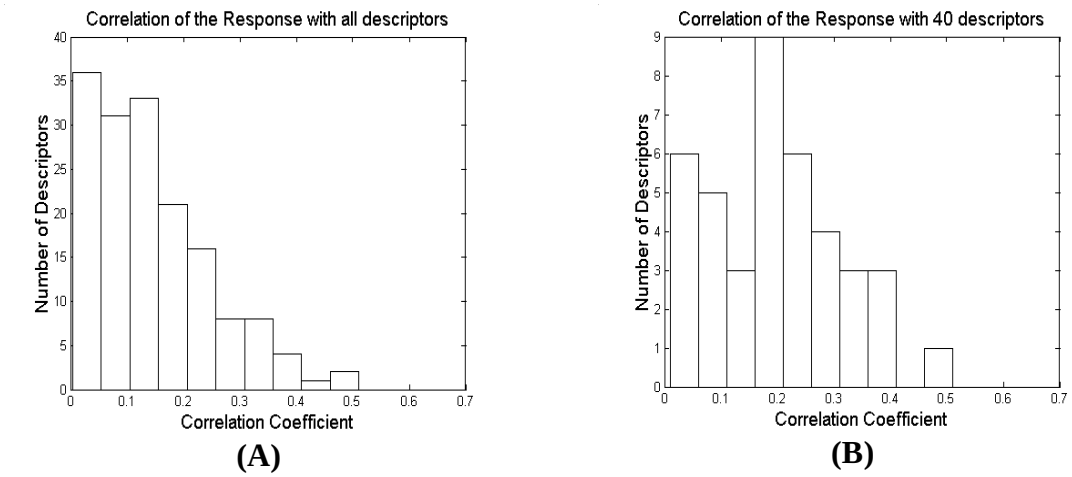


Figure 8.14 (A) Histogram of the Correlation of the Response Variable with All 160 Descriptors **(B)** Histogram of the Correlation of the Response Variable with the 40 Selected Descriptors by GAFEAT ($\alpha = 0.55$)

The feature subset with 40 descriptors selected by GAFEAT is further pruned down by using neural network sensitivity analysis explained in section 6.5. Sensitivity analysis was able to reduce the number of features from 40 to 31 and slightly improved quality of the model. Neural network sensitivity analysis was applied to the HIV_{rt} dataset with 160 descriptors and selected 35 descriptors out of 160. For comparison purposes, the HIV_{rt} dataset with all 160 descriptors was used to construct the neural network model.

The training set size is set to 58 molecules, leaving 6 molecules for the validation set. In order to compare the results of the models, the MLP is trained and tested for 100 bootstraps and average error rate of all bootstraps is reported in terms of q^2 and Q^2. Table 8.9 summarizes those results.

Table 8.9 Comparisons of Methods

METHOD	Number of Selected Descriptors	MLPs Predictive Results based on 100 bootstraps	
		q^2	Q^2
NO FEATURE SELECTION Using all 160 descriptors	160	0.419	0.440
GAFEAT Using all 160 descriptors	40	0.364	0.380
NN SENSITIVITY ANALYSIS Using 40 descriptors selected by GAFEAT	31	0.359	0.377
NN SENSITIVITY ANALYSIS Using 160 descriptors	35	0.312	0.340

Figure 8.15 shows the result of the predictive modeling for the dataset with 31 features selected by neural network sensitivity analysis using 40 features selected by GAFEAT. The prediction variance is also shown in this figure for each molecule.

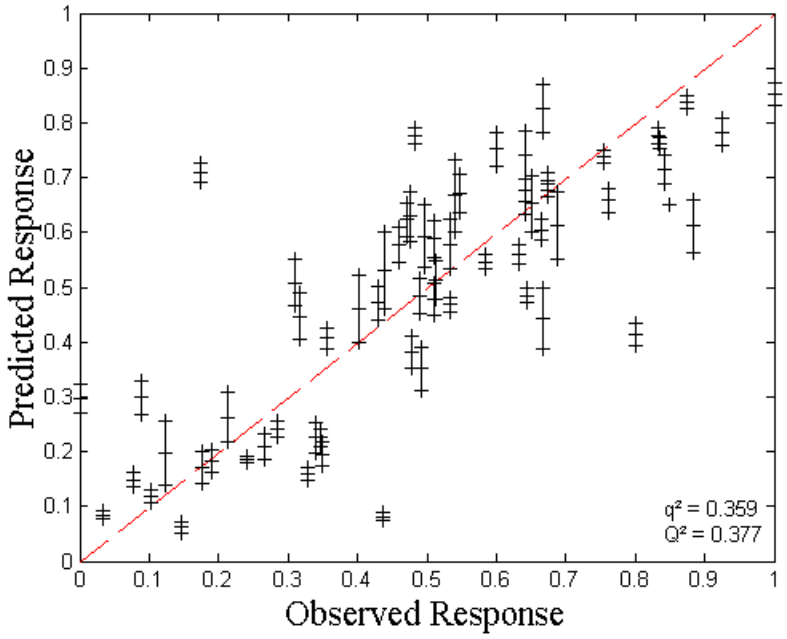


Figure 8.15 MLP Prediction Results for the Dataset with 31 Features Selected by Neural Network Sensitivity Analysis using 40 Features Selected by GAFEAT

8.7.3 Performance of GAFEAT on Feature Selection

In this section, GAFEAT is applied for feature selection in order to build a good predictive MLP model. For this purpose, two Caco-2 cell permeability datasets are employed. These datasets are explained in section 7.4. First dataset is Caco-2 cell permeability data for 27 structures [175]. These 27 structures will be modeled separately according to two response variables: LogPC and Logpapp. The second one contains 48 compounds and is used to model LogPC [176]. Each dataset initially contains 780 descriptors (274 TAE descriptors [159], 396 PEST descriptors [160, 161], and 110 selected MOE descriptors that are related with pharmacophore, shape and volume).

The next step was to discard descriptive features, which contained constant values. This reduced the number of descriptive features to 715 and 703 for dataset I (27 molecules) and dataset II (48 molecules), respectively. GAFEAT with unique list representation was used to select 20 features from both datasets. The parameter settings of GAFEAT for both datasets are the following:

Population size : 200	*Number of features to be selected* : 20
Maximum number of generation : 1000	*Crossover probability* : 0.90
Mutation probability : 0.02	*Inter-correlation penalty* : 0.55
Death penalty : 0.95	

The selected 20 features from both datasets are presented in Table 8.10.

Table 8.10 20 Features Selected by GAFEAT for Caco-2 Datasets

20 FEATURES SELECTED BY GAFEAT FOR Caco-2 DATASETS		
DATASET I (27-molecule)		**DATASET II (48-molecule)**
Response: LogPC	Response: LogPapp	Response: LogPC
Feature Label	**Feature Label**	**Feature Label**
ABSDRN6	ABSDRN6	ABSDRN6
ABSEPMIN	SIEPMIN	DGNB03
PIP2	PIP2	DGNB52
PIP16	PIP16	DRNB53
KB44	KB11	GB53
KB54	FUKB14	KB04
FUKB14	PIPB53	KB35
PEOE_VSA+3	BNPB31	KB52
PEOE_VSA_FHYD	PEOE_VSA+3	PIPB03
PEOE_VSA_FPNEG	PEOE_VSA_FHYD	PIPB04
PEOE_VSA_FPPOS	PEOE_VSA_FPNEG	PIPB34
PEOE_VSA_PPOS	PEOE_VSA_FPPOS	PIPB35
Q_VSA_FPNEG	PEOE_VSA_PPOS	BNPB40
pmiY	Q_VSA_FPNEG	PEOE_VSA-1
a_don	pmiY	PEOE_VSA_FPOS
SlogP	a_don	RPC+
SlogP_VSA0	SlogP	dipoleY
SlogP_VSA6	SlogP_VSA6	a_don
SMR_VSA4	SMR_VSA4	vsa_pol
SMR_VSA7	SMR_VSA7	SlogP_VSA0

A standard feed-forward multi-layered perceptrons trained with the back-propagation algorithm is used to build a predictive model. The artificial neural networks have two hidden layers and are oversized (the hidden layers for this study contained 13 and 11 neurons respectively). Training phase of the neural networks was halted at 0.105 with early stopping by policy. The training set sizes are set to 24 and 43 molecules, leaving 3 and 5 molecules for the validation sets for Dataset I and Dataset II, respectively. In order to compare the results of the models, the MLP is trained and tested for 100 bootstrap samples and average error rate of all bootstraps is reported in terms of q^2 and Q^2. The predictive results of MLPs for Caco-2 datasets based on 100-bootstrap are presented in Table 8.11. According to these results GAFEAT selects relevant features and helps us to build a better predictive model compared to using all features.

Table 8.11 Predictive Results of MLPs for Caco-2 Datasets based on 100-bootstraps

Method	DATASET I (27-molecule)				DATASET II (48-molecule)	
	Response: LogPC		Response: LogPapp		Response: LogPC	
	q^2	Q^2	q^2	Q^2	q^2	Q^2
Without Feature Selection (Using all features)	0.575	0.582	0.554	0.558	0.494	0.494
With Feature Selection (Using 20 features selected by GAFEAT)	0.174	0.179	0.158	0.165	0.253	0.256

8.7.4 Conclusions

One conclusion that can be easily drawn from the experimental results is that GAFEAT selects a good initial subset of features and improves the predictive quality of the learning algorithms. The predictive ability of the learning algorithms heavily depends on the selection of the appropriate value for the inter-correlation penalty factor (a). According to computational results, if a right alpha value is selected, GAFEAT can select relevant features and help to build a better predictive model compared to using all features. It has been found that the low values as well as high values for inter-correlation penalty factor deteriorate the prediction error of the learning algorithms, and a good value for a lies between 0.40 and 0.80.

Neural network sensitivity analysis is also useful tool to prune down the features selected by GAFEAT and to lower the prediction error. If we compare GAFEAT and sensitivity analysis as a stand-alone method, sensitivity analysis performs better than GAFEAT. One of the reasons is that sensitivity analysis can be thought of as a wrapper approach and takes the biases of the model into consideration. The other reason is that sensitivity analysis is an iterative method that starts from full set of features and drops insignificant features iteratively. On the other hand, one of the advantages of GAFEAT is

that it scales only with the number of features. Therefore, GAFEAT does not require extra computing cost if the number of the data points in a dataset increases. The advantage of GAFEAT over the sensitivity analysis is that it requires less computation time. Since the most of the wrapper methods require more computation time with respect to the number of descriptors and data points, GAFEAT can be used as a filter to reduce the number of descriptors for the wrapper methods.

CHAPTER 9

GAFEAT-PLS: Genetic Algorithms with Partial Least Squares Regression for Feature Selection

Partial least squares projections to latent structures (PLS) has a useful method for modeling highly multidimensional scientific datasets with collinear features [103, 180-182]. PLS regression is a Principal Component Regression (PCR) based on latent variables, in which the direction of the latent variables is slightly shifted from the PCR solution to obtain optimal correlation between the response variables and independent variables [103].

In the past, PLS regression was considered to be almost insensitive to noise; therefore, there was a common acceptance that no feature selection was necessary to build a better predictive model [183]. Today, it has been widely accepted that a feature selection has some advantages. Although PLS is a well-working method to model highly multidimensional and collinear datasets, the interpretation and understanding of the predictive model and its results are more difficult [182]. Feature selection can also help to built a better predictive PLS model with fewer features [103, 183].

In this chapter, an error measure for PLS regression is integrated as a cost function inside the genetic algorithm with the unique list representation developed in section 8.3.3 in order to perform feature selection. First of all, detailed information about PCR and PLS is presented. Second, a literature review related to feature selection with evolutionary algorithms and PLS is reviewed. In later sections, the details of the proposed

Genetic Algorithm with Partial Least Squares (GAFEAT-PLS) for feature selection and its performance on two QSAR datasets (the Lombardo and HIV_{rt} will be presented.

9.1 Data Compression

The basic idea behind a 'data compression' or 'rank reduction' is that the information in the many observed data variables $x = (x_1, x_2, ..., x_K)$ can be compressed into a few underlying **latent** variables (also called components, scores, regression factors or just factors) $t_1, t_2, ..., t_A$ [184]. This relationship can be mathematically expressed as

$$(t_1, t_2, ..., t_A) = h_1\{(x_1, x_2, ..., x_K)\} \tag{9.1}$$

and these latent variables can be used as regressors to build a regression model with response variable **y**.

$$y = h_2\{(t_1, t_2, ..., t_A)\} + e \tag{9.2}$$

In equation (9.2), **e** represents those contributions to **y**, which cannot be explained by latent variables. The **A** latent variables (**A** < **K**) are assumed to represent the systematic variation in the original data variables, which are important for predicting the response variable **y**. The functions $\mathbf{h_1}$ and $\mathbf{h_2}$ in equations (9.1) and (9.2) can be combined together to build a predictive regression model for **y**.

$$\hat{y} = h_2\{h_1(x)\} \tag{9.3}$$

Data compression helps to simplify model-building phase by reducing the number of model parameters and to solve the collinearity problem by guaranteeing an invertible matrix in calculation of regression coefficients [104]. It can also simplify the interpretation of the results since a few latent variables can reveal the main relationship between large numbers of original variables. On the other hand, data compression has

some disadvantages. Some useful information can be contained in the discarded latent variables and/or a retained latent variable may not have predictive information [104, 184]. The interpretation of the results may be cumbersome since the latent variables are some combinations of the original variables. Many different methods are available for data compression. Principal Component Regression (PCR) and Partial Least Square (PLS) Regression are well-known methods.

9.1.1 Principal Component Analysis (PCA) and Principal Component Regression (PCR)

Principal Component Analysis (PCA) is a powerful visualization tool and is widely used in explanatory data analysis and data compression (or rank reduction). PCA is based on the fact that any set of M variables can be transformed to a set of M orthogonal variables [104, 126].

$$\mathbf{X} = \mathbf{T}\mathbf{P}^{T} \tag{9.4}$$

The symbol ^{T} stands for the transpose of the matrix. **X** is the data matrix with N rows and M columns, which correspond to the number of samples and independent variables, respectively. **T** is the score matrix with N rows and M columns form the so-called principal components of **X** and **P** is the loading matrix with M rows and N columns. The columns of **P** are made up of the eigenvectors of $\mathbf{X}^{T}\mathbf{X}$. The elements of the principal components are the axes of the principal component line. The score matrix **T** is the projection of the respective points on the principal component lines.

PCA is a classical statistical method and mostly used in data analysis where the **X** variables are expected to be collinear. The collinearity means that the **X** data matrix has

some dominating types of variability that explain most of the available information. PCA is based on the concept of *variation* and built on the assumption that variation implies information that might be classified as either relevant or irrelevant. Then, the purpose of the PCA is to express dominant information in the data matrix $\mathbf{X} = \{x_m, m = 1, 2, \ldots, M\}$ by a lower number of variables (principal components of **X**) $\mathbf{T_A}=\{t_1, t_2, \ldots, t_A\}$ where $A < M$. [184]. Then, redundancy and smaller noise variability are removed. The PCA provides a way to reduce the dimensionality of the data in such a way that linear combinations of **X** variables account for maximal amount of variations. In this situation, the first A columns of the score matrix **T** and the loading matrix **P** are considered.

$$\hat{X} = T_A P_A^T \tag{9.5}$$

where $\hat{X}$ is an estimation of **X**.

Principal Component Regression (PCR) is obtained by regressing the dependent variable **Y** on the score matrix **T**. In PCR, the variables of the **X** are replaced by those of **T**, which are orthogonal to each other and span the same multidimensional space of **X** if all principal components are combined [104].

$$\hat{X} = T_A P_A^T \tag{9.6}$$

$$T_A = XP_A \tag{9.7}$$

$$Y = T_A B + E \tag{9.8}$$

One of the advantages of the PCR is that it solves the ill conditioning in the matrix inversion due to multi-collinearity problem in the calculation of the regression coefficients (**B**) by producing orthogonal variables. A second advantage is that eliminating some of the principal components can cause some random error elimination. A key important advantage of the PCR is that, in contrast to MLR, it can also be applied

to and give good results when there are more X-features than data points, as long as the predictive information is in the first few principal components (eigenvectors) [184].

9.1.2 NIPALS Algorithm

The NIPALS (**N**onlinear **I**terative **PA**rtial **L**east **S**quares) algorithm is the most commonly used method for calculating the principal components for a dataset [151]. The algorithm extracts one component a time and finds A principal components without calculating all the eigenvectors [104]. It extracts t_1 and p_1^T from the data matrix X. The outer product of $t_1 p_1^T$ is subtracted from X in order to get residual the E_1. Then, E_1 is used to calculate t_2 and p_2^T.

$$E_1 = X - t_1 p_1^T, \quad E_2 = E_1 - t_2 p_2^T, \quad E_3 = E_2 - t_3 p_3^T, \ldots \tag{9.9}$$

The implementation of the NIPALS algorithm is discussed more detail in [104, 184, 185]. The complete NIPALS algorithm is summarized in Appendix A.

9.1.3 Partial Least Squares (PLS) Regression

Partial Least Squares (PLS) is a general method of handling regression problems. This method allows relationships between many blocks of data to be characterized and modeled [105]. Also, this method can model data with strongly correlated and/or noisy or numerous independent variables and several response variables [180]. The simplest type of application of PLS methods is the PLS regression. The PLS regression is a generalization of Multiple Linear Regression (MLR). The results of the PLS regression are analogous to the MLR. In addition, the PLS regression also produces a set of plots

(scores and loadings) that provide information about the correlation structure of the variables and the structural similarities/dissimilarities between the compounds [180].

Both PCR (explained in section 9.1.1) and PLS regression are factor based regression methods that produce factor scores as linear combinations of the original X variables. On the other hand, PLS and PCR differ in their ways of extracting factor scores. In the PLS regression the latent variables are extracted both to model X and to correlate with Y, which is contrast to PCR, in which the latent variables only model X [105].

In the PLS regression, a relationship is modeled between a response variable matrix Y and a data matrix X. The modeling procedure begins with scaling the Y and X columnwise with mean zero and variance one. The PLS model is built on the properties of the NIPALS algorithm explained in section 9.1.2. There are many variants of the NIPALS algorithms that normalize or do not normalize certain vectors [184].

The following explanation of the PLS regression is taken from Geladi and Kowalski [104, 186]. A PLS model consists of outer relations (X and Y matrices individually) and an inner relationship between both matrices. The outer relation for the X data can be written as:

$$X = T\ P^T + E = \sum_{h=0}^{A} t_h p_h^T + E \qquad (9.10)$$

where A is the number of latent variables. The outer relation for the Y data also can be written in the same way:

$$Y = U\ Q^T + F^* = \sum_{h=0}^{A} u_h q_h^T + F^* \qquad (9.11)$$

The regression part of the PLS is an inner relation, which is described as:

$$\hat{u} = b_h t_h \tag{9.12}$$

where $b_h = u_h^T t_h / t_h^T t_h$ and the "hat" indicates that the vector is an estimated one. The mixed relation can be written as:

$$Y = T\ B\ Q^T + F \tag{9.13}$$

where $\|F\|$ has to be minimized with the condition that $\|E\|$ in equation (9.10) is minimized.

It is necessary to introduce weighting (**w**) to get orthogonal scores as in PCR. The PLS model is obtained by using the residuals after each dimension, which can be written as:

$$E_h = E_{h-1} - t_h p_h^T; \quad X = E_0 \tag{10.14}$$

$$F_h = F_{h-1} - b_h t_h q_h^T; \quad Y = F_0 \tag{10.15}$$

The complete PLS algorithm is given in the Appendix B.

9.1.4 Optimal Number of Latent Variables

An optimal number of latent variables (PLS regression components) (A_{opt}) needs to be estimated. In general, A_{opt} is chosen as the model rank that minimizes some criterion (e.g., prediction error on a validation set) for the different models A = 3,4,5,…,A_{max}. There is a high risk for overfitting when working with dataset with large number of inter-correlated features. Generally, this criterion is calculated with cross-validation to test the predictive ability of the model, and the best one is chosen [181]. When several models, e.g., A = 4, 5, 6 perform about the same based on the criterion used, a lower-dimensional model is preferred [187].

The HIV_{rt} and Lombardo datasets are modeled with PLS regression with a different number of latent variables under the leave-one-out cross-validation. The HIV_{rt} datasets has 64 molecules and 230 descriptive features and the Lombardo datasets has 62 molecules and 309 descriptive features (the datasets are pre-processed using StripMiner^{TM} program). The results of the leave-one-out cross validations for different number of latent variables for the HIV_{rt} and Lombardo datasets are presented in Tables 9.1 and 9.2, respectively. The plot of the prediction errors for the both datasets in terms of Q^2 versus the number of latent variables is shown in Figure 9.1. It is apparent from Figure 9.1 that the optimal number of latent variables for both datasets is four (lowest leave-one-out prediction error).

Table 9.1 Leave-One-Out Cross-validation Results of the PLS Regression Model with Different Latent Variables for the HIV_{rt} Datasets

HIV_{rt} DATASET				
# of latent Variable	r^2	R^2	q^2	Q^2
3	0.7682	0.7682	0.3960	0.3974
4	0.8044	0.8043	0.3396	0.3400
5	0.8517	0.8517	0.3417	0.3464
6	0.8822	0.8822	0.3397	0.3494
7	0.9172	0.9170	0.3575	0.3790
8	0.9384	0.9381	0.3978	0.4283
9	0.9590	0.9585	0.4024	0.4522
10	0.9622	0.9617	0.4302	0.5005

Table 9.2 Leave-One-Out Cross-validation Results of the PLS Regression Model with Different Latent Variables for the Lombardo Datasets

LOMBARDO DATASET				
# of latent Variable	r^2	R^2	q^2	Q^2
3	0.8760	0.8760	0.2926	0.2932
4	0.9158	0.9157	0.2855	0.2868
5	0.9497	0.9496	0.2855	0.2910
6	0.9622	0.9621	0.2911	0.2992
7	0.9693	0.9692	0.3035	0.3170
8	0.9766	0.9765	0.3221	0.3464
9	0.9811	0.9809	0.3426	0.3805
10	0.9851	0.9850	0.3618	0.4108

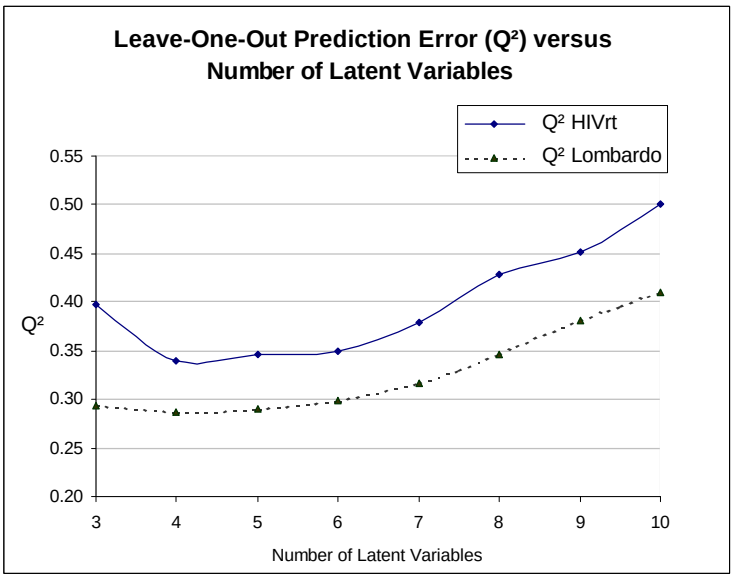


Figure 9.1 Leave-One-Out Prediction Errors of the PLS Regression Models for the HIV_{rt} and Lombardo Datasets versus Number of Latent Variables

9.1.5 Nonlinear Partial Least Squares (PLS) Regression

In general, real datasets (e.g. QSAR datasets) exhibit significant nonlinear characteristics, which cannot be properly modeled by linear regression methods. Non-linearity can be handled either using models with a nonlinear function in the regression step or making relationship between X and Y data linear and using ordinary least squares methods [188].

In the first approach, the development of a nonlinear PLS model is to modify the inner relationship of the linear PLS regression by introducing a nonlinear function (a quadratic polynomial, a spline function) that relates the output scores u to input scores t by keeping the X and Y data intact [189-191]. However, these methods require a number of parameters (e.g. degree of the polynomial and spline knots placement) to be

determined, which is difficult and adds additional complexity to the problem. These methods also easily overfit the data since they are very flexible [192].

In the second approach, a new expanded X dataset is constructed by including higher order terms (e.g. quadratic terms, cross-product terms) of the original variables of the X data. Then, a linear PLS regression is fitted into the new expanded X data and Y data. This way, a nonlinear variant of the linear PLS algorithm can be produced by integrating nonlinear variables within the linear framework [191]. The disadvantage of this method is that if dataset has K features and quadratic nonlinearities are included, the number of cross-product term will be $\frac{K(K-1)}{2}$, which increases rapidly when K increases. Therefore, it is clearly seen that the full expansion of the features is not suitable for QSAR datasets since a typical QSAR dataset for predicting an activity of interest is characterized by a large number of descriptive features (300-1000).

Berglund and Wold [192] proposed a simple way to develop nonlinear PLS models within the linear PLS framework. Since PLS produces a model in which each latent variable is a linear combination of the original X variables, if Y is a nonlinear function of these latent variables; then, the X data must be expanded in such a way that the square and cross-product terms of the latent variables exist in the model. They show that by simply adding squared variables of the X data, both the square and cross product terms of the latent variables are implicitly included in the resulting linear PLS model. This method, which is called as INLR (**I**mplicit **N**onlinear **L**atent variable **R**egression), works well when X data is well modeled by a projection model ($X \approx T P^T$) and for continuous rather than binary features. Therefore, if a latent structure is present in the X,

the cross-product terms of the variables of the X can be excluded in the polynomial expansion.

The principle of the INLR taken from [192] is the following:

Let **Y** be a nonlinear function in **T** (in the simplest case, polynomially quadratic):

$$Y_{im} = \{T\ Q^T\}_{im} + \sum_{a=1}^{A} \sum_{b=a}^{A} t_i t_{ib} d_{abm} + f_{im} \qquad (9.16)$$

The relationship in equation (9.16) can be modeled by estimating the coefficients (d_{abm}) and the latent variables t_a, t_a^2, and $t_a t_{a+1}$. On the other hand, it can be shown that this relationship can be modeled by the expanded predictor matrix (XX^2) that includes only original variables of the X data and their squares. Let's assume $X \approx T\ P^T$, the squared term can be written as

$$x_{ik}^2 = \sum_{a=1}^{A} \sum_{b=a}^{A} t_{ia} t_{ib} p_{ak} p_{bk} + e_{ik} \qquad (9.17)$$

Equation (9.17) shows that by expanding X with only the squared terms, both the quadratic and the cross-product terms of latent variables are implicitly included in the PLS model of the X block. This can be shown on a simple example of a X matrix with rank two (A=2):

$$X = T\ P^T = t_1 p_1^T + t_2 p_2^T \qquad (E = 0) \qquad (9.18)$$

Let's consider only the first element of X, x_{11}, then $x_{11} = t_{11} p_{11} + t_{21} p_{21}$. The squared term of x_{11} is calculated as

$$(x_{11})^2 = (t_{11} p_{11} + t_{21} p_{21})^2 = (t_{11} p_{11})^2 + (t_{21} p_{21})^2 + 2(t_{11} p_{11} t_{21} p_{21}) \qquad (9.19)$$

Equation (9.19) is equal to Equation (9.18). The same approach is also valid for the cubic extension of X data. This is a logical extension of the ordinary linear PLS, if there is a latent structure in the data that is related to the response variables [188]. The underlying assumptions are that the X data has a latent structure and that there are enough X-variables to support the increased dimensionality of the expansion. Therefore, the number of latent variables A must be substantially smaller than the number of variables and the initial number of variables (K) should be larger than three times the rank of X [192]. When comparing a number of different approaches to non-linear PLS modeling (PLS with non-linear inner relationship [190], neural network PLS [193, 194]) with INLR [192], it has been pointed out that all except INLR are somewhat cumbersome to use and have tendency to be too flexible, causing overfitting, especially in small datasets [195].

9.2 Feature Selection with Evolutionary Algorithms and PLS Regression

An evolutionary approach, the MUSEUM (**MU**tation and **SE**lection **U**ncover **M**odels) algorithm, was proposed in [101, 103] for feature selection in the regression and PLS analyses for QSAR datasets. The MUSEUM algorithm starts from an arbitrary regression model and adds or drops features to or from this model based on a mutation mechanism. The fitness of a model is defined by a certain criterion, e.g. the standard deviation or the Fischer significance value *F* of the regression model. If a mutated model has higher fitness than that of its parent, it is taken as a new breeding model that will be further mutated by feature additions or eliminations. If no improved model is obtained after a predetermined number of mutation operations, the mutation rate is increased. If a

better model still cannot be found, the current best model is accepted as an intermediate result. In the next step, in order to check whether all feature combinations have been considered by mutation operations, all features not included and included by the current best model are systematically added and eliminated, one at a time. If an improved model is obtained, the MUSEUM algorithm starts again with this improved model. On the other hand, if there is no improved model found, all features included by the model are tested for significance at 95% confidence level and insignificant features are eliminated.

Leardi et al. proposed a modified binary genetic algorithm for feature selection in regression and PLS analyses for QSAR datasets [7, 183, 196]. In a GA applied to a dataset with *k* features for feature selection, the structure of a chromosome consists of binary string with the size of *k* in which each gene is represented a single bit (0 = corresponding feature is absent, 1 = corresponding feature is present). The objective of the GA is to find a feature subset that maximizes the percentage of predicted variance (R^2). As an example, an individual could be 0010011001 for a dataset with 10 features. The fitness of this individual will be the variance predicted by the PLS model computed by taking into account features 3, 6, 7, and 10. The following modifications were applied to the binary GA to take into account some peculiarities of the feature selection problem:

i) An initial population is formed by individual corresponding to subsets of only a few features in each. At stage of the creation of the initial population, the probability of having '1' is much lower than that of having '0'.

ii) Since the fitness function requires a full multivariate analysis (MLP, PLS, etc.) to obtain a predicted value using cross-validation, the time required by the fitness

evaluation can be excessive. For this reason, the population size of the GA is kept as low as possible (30 individuals).

iii) A high degree of elitism is introduced to GA by allowing parents and their children coexist if their fitness's are higher respect to population.

iv) The best individual containing *C* number of features is added into next generation regardless of its fitness value unless an individual with a same or lower number of features give a better fitness value. This rule forces the GA to have solutions containing as few features as possible since a feature subset with a few features leads to an easier mathematical model and sometimes to lower the computational cost. At the end of the run, the evolution of the fitness as a function of the number of selected features can be tracked in the individuals of the last population.

v) The GA is hybridized with stepwise selection. A backward stepwise selection is performed on the best individual of each GA generation and resulting individual is considered as an offspring.

Hesegawe et al. employed the modified GA proposed by Leardi et al. to obtain a PLS model with high internal predictivity using a small number of features in QSAR datasets [197]. They applied the GAPLS to the inhibitory activity of calcium channel antagonists. As a result, they selected the features strongly contributing to the inhibitory activity and estimated the structural requirements for the inhibitory activity in an effective manner.

Genetic Partial Least Square (GPLS) was proposed to construct QSAR models by Duns et al. [99]. The GPLS method uses a GA to select appropriate basis functions (features) to be used to model a QSAR data. The initial models are generated by

randomly selecting a number of features using the user-specified basis function type, and then, constructing the models from random sequences of these basis functions. In GPLS, individuals are a series of basis functions. The lengths of the series (individuals) are predetermined. For each individual, a QSAR model is construct by using PLS regression to generate the regression coefficients for the basis functions. The fitness of the individuals (models) is rated using a modified form of Friedman's 'lack of fit' (LOF) [198]. LOF is defined as in equation 10.5

$$\mathrm{LOF} = \frac{\mathrm{LSE}}{\left[1 - \frac{c\,(d+1)}{N}\right]^2} \qquad (10.5)$$

The terms in the equation 10.5 are defined as follows. *LSE* is the least-square error, *N* is the number of compound in the dataset, *d* is a smoothing parameter, and ***c*** is the number of basis functions (independent variables) in the model. The objective of the GPLS is to minimize LOF since the smaller the LOF is the better model. GPLS uses classical one point uniform crossover. The parents for crossover are chosen based on the inverse of their LOF scores.

9.3 The Proposed GAFEAT-PLS

In the Genetic Algorithm proposed in this thesis with Partial Least Square Regression (GAFEAT-PLS), the PLS regression is integrated as a cost function into the GA with unique list representation developed in section 8.3.3. Given a dataset containing T features, each chromosome represents a legal subset containing N features. In this representation, a chromosome is an integer array with size N, where N is the predetermined number of features to be selected out of total T features. Each gene

represents the corresponding feature in the dataset. A flow diagram describing the procedure of GAFEAT-PLS is presented in the Figure 9.2.

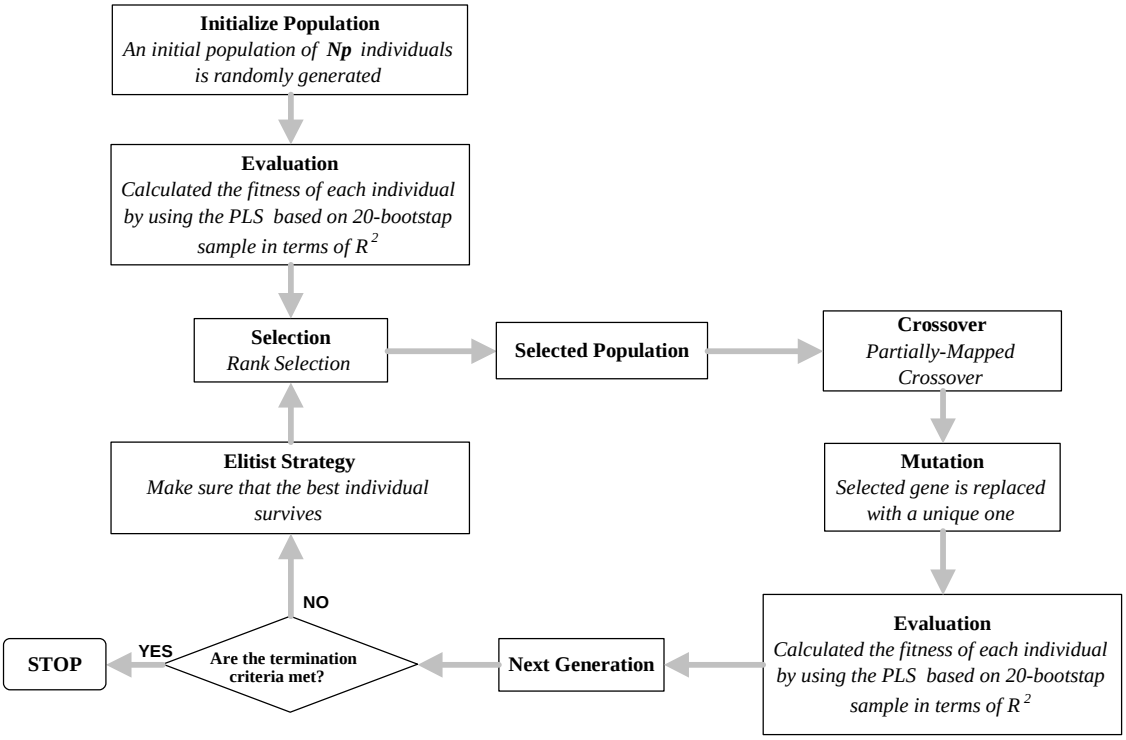


Figure 9.2 Flow Diagram of GAFEAT-PLS Algorithm

9.3.1 Creation of the Initial Population

The initial population is created randomly as explained in section 8.3.1. In the initial population, all features in each individual are guaranteed to be different (i.e., there are no duplicated feature). Although the optimal number of chromosomes (individuals) depends on the number of features in the dataset under scrutiny, a population size of 100 is sufficient large for GAFEAT-PLS to converge for up to 1500 features for small datasets.

9.3.2 Evaluation of the Fitness

The fitness of each individual (chromosome) is evaluated by the predictivity of the PLS model derived from the feature subset represented by the corresponding individual. R^2 is used as a yardstick for estimating the prediction ability of the PLS models (see section 6.4). GAFEAT-PLS calculates the fitness of each individual based on a certain number of bootstrap samples. In this implementation, the (**n**-**h**) compounds (samples) for the training set are drawn without replacements, where **n** is the number of compounds in the data set and **h** is the number of samples left in the holdout set. The number of samples left in the holdout set is generally around 10 percent of the total number of samples in the dataset. Note that this holdout set is neither test nor validation set and is never used for any purposes such as preventing over-fitting. There are no overlap between training and holdout sets for a bootstrap sample. After the PLS model is constructed by using the training set, R^2 is calculated on this training set. This process is repeated 20 times and the fitness of each individual in the population is obtained based on the average R^2 for 20 bootstrap samples. The fitness of the individual **k** is given by

$$\text{Fitness}_k = \frac{1}{20}\sum_{i=1}^{20} R_i^2 .$$

The objective of GAFEAT-PLS is to find the individuals *maximizing* this fitness function. The values of the fitness function will be between $-\infty$ and +1, since the values for R^2 lie between $-\infty$ and +1.

In order to show how GAFEAT-PLS calculates the fitness for individuals, let's assume that the objective of GAFEAT-PLS is to select 10 features from a dataset with 62 molecules and 694 descriptive features. Figure 9.3 graphically illustrates this example. For each individual, 56 molecules are randomly drawn without replacement from the

total of 62 molecules to construct a training set that contains only the features represented by the corresponding individual. A PLS regression is fitted this training set and the goodness-of-fit of this model is measured in terms of R^2 statistic. This process is repeated 20 times and the final fitness of the individual is calculated based on the average of these R^2 of 20 bootstrap training sets.

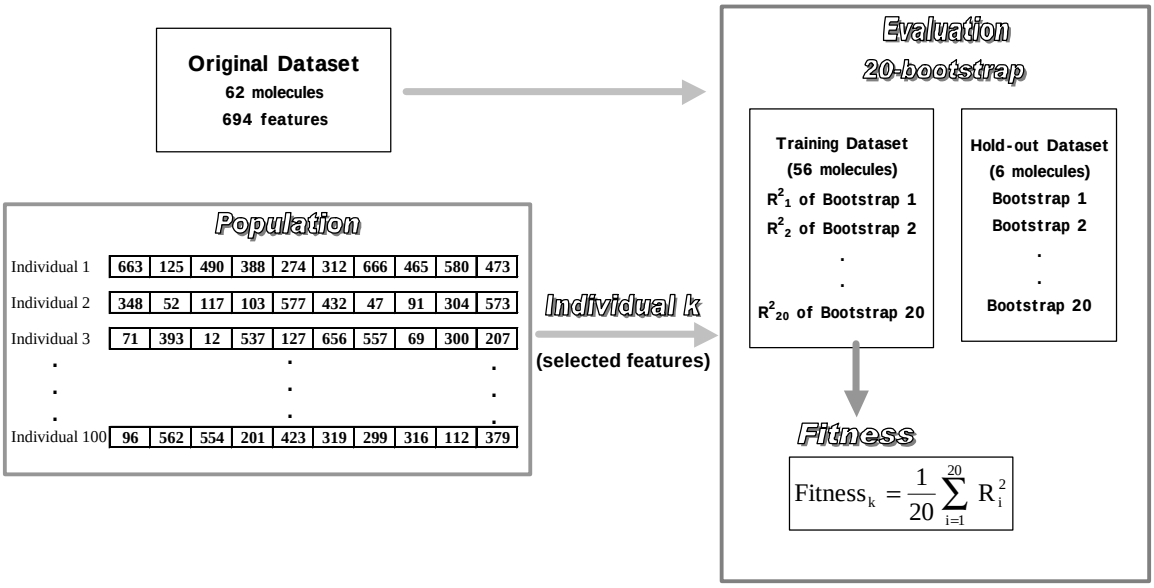


Figure 9.3 Illustration of the Fitness Evaluation in GAFET-PLS Algorithm

9.3.3 Selection Mechanism

Selection is the process of deciding on which individuals will survive for the next iteration in the algorithm. GAFEAT-PLS employs a rank-based selection, which means that only the rank ordering of the fitness of the individuals within the current population determines the survival probabilities. There are many methods to assign the survival probability based on ranking (see [8, 199]). GAFEAT-PLS assigns new fitness values to the individuals of the population based on their ranking using the following function:

$$\text{Fitness}_{(\mathbf{rank})} = (\text{pop\_size} - \text{rank} + 1)^{\theta}$$

This function returns a new fitness value of an individual ranked in position in **rank**, where **pop_size** is the number of individuals in the population and **rank** = 1, 2, 3, …, **pop_size** (**rank** = 1 means the best individual and **rank** = **pop_size** means the worst individual in the population based on the actual fitness). Individuals for the next generation are selected proportionally to their new fitness values rather than the actual objective function values. q is a user-defined parameter that allows the users to influence the selective pressure. The acceptable range for the q parameter is between 0 and 10. q = 0 means that there is no selective pressure and all individuals have the same survival probability regardless of their actual fitnesses. If value of θ increases the survival probability of the better individuals increases. Figure 9.4 shows the ranking versus the survival probabilities for different selective pressures (q) of 0.0, 1.0, 1.8, 5.0, and 10.0 for a population containing 100 individuals.

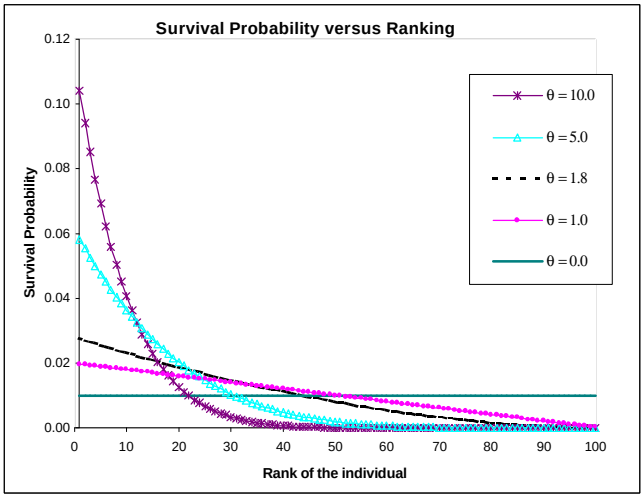


Figure 9.4 Survival Probability for a Population Size of 100 individuals versus Ranking for Different Selective Pressure (q) Values

9.3.4 Crossover and Mutation

In the proposed GAFEAT-PLS, after the rank-based selection (or sampling) mechanism, a recombination (crossover) operator is applied to the individuals selected based on a predefined crossover probability. GAFEAT-PLS employs a sexual (two parents) reproduction, namely partially mapped crossover (PMX) explained in section 8.3.4 so that the resulting two offspring contain maximal characteristics from both parents. The proportion of the population that undergoes crossover during a generation is determined by the crossover probability. For example, if the probability of crossover is 90 percent, we expect that on average 90 percent of the population will undergo crossover and these individuals are paired off as parents.

After the crossover operation, mutation follows. The mutation operator randomly chooses a gene position in the chromosome and tries to exchange it with a unique random value. Here, the unique value means that the mutated gene position must have a different value (indicating a new feature) than all values for a given individual. If a unique value cannot be found a certain number of trials then the parent chromosome is copied without any change.

9.3.5 Stopping Criteria

A key issue is deciding when to stop evolving solutions with a genetic algorithm. The proposed GAFEAT-PLS uses two termination criteria: the allowed maximum number of generation and the early stopping. Once one of them is satisfied, GAFEAT-PLS stops running and reports the last population and the fittest individual of the GA run.

The allowed maximum number of generation must be large enough to allow GAFEAT-PLS to converge.

The second termination criterion is the early stopping. The fitness function of GAFEAT-PLS explained in the Section 10.6.2. The objective of GAFEAT-PLS is to find the solutions (feature subsets) maximizing the fitness function defined by

$$\text{Fitness}_k = \frac{1}{20}\sum_{i=1}^{20} R_i^2$$

The values of the fitness function will be between $-\infty$ and 1. GAFEAT-PLS stops running if an individual with fitness value equal to 1 has been found. However, the threshold value for the fitness can be specified as a value less than 1 in order to halt a GAFEAT-PLS run early.

9.4 Validation of GAFEAT-PLS

The main goal of the feature selection is to select a subset of the original features such that the resulting model can perform well on unseen future data points (compounds) [200]. The commonly used validation strategy for the feature selection consists of the following two steps:

Step 1. The selection of features by using all the data points,

Step 2. The model obtained with the selected features is validated under a validation scheme (cross-validation, bootstrapping, etc.).

Because the selection of the features and model validation are performed on the same data points, the predictive model is only partially validated [200, 201]. Figure 9.5 illustrates the classical (partial) validation approach to feature selection. It has been reported that the prediction quality or ability of a predictive model validated in this way

is optimistically biased, since the data points used for the feature selection is also employed in the model building and validation step [6, 196, 200, 201].

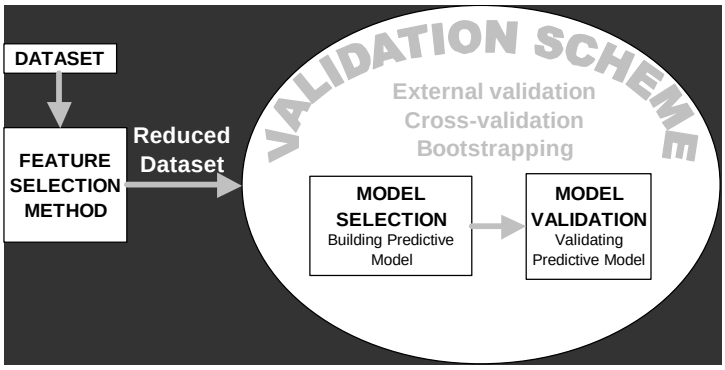


Figure 9.5 Classical (Partial) Validation Approach to Feature Selection

It has been pointed out by many researchers that in order to perform a full validation, the feature selection, model selection, and model validation must be performed under the same validation scheme [6, 196, 201]. The full validation approach to feature selection is illustrated in Figure 9.6.

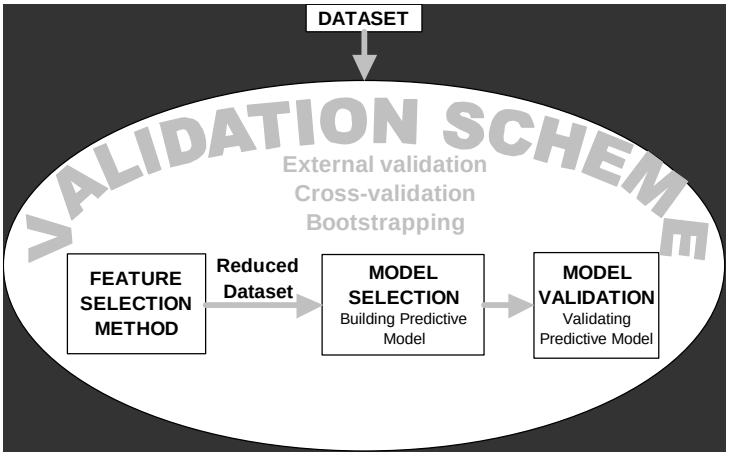


Figure 9.6 Full Validation Approach to Feature Selection

Any QSAR model needs to be properly validated prior to use for predicting biological activities of new untested molecules. QSAR models have a theoretical foundation as semi-empirical analogy models such that they usually have local validity, and that they can only predict molecules that are chemically and biologically similar to those of the training set [202]. Therefore, the most important step in building a sound and robust OSAR model is the selection of an informative and representative training set. The validation process, in general, is a very important step and must be carried out in the right way. The most commonly used methods for a model validation are the use of an external validation (test) set and cross-validation. A randomization test is also commonly used for the validation of QSAR models [124]. These validation methods are explained in more detail in the following sections.

9.4.1 External (Independent) Validation Set

The most reliable method for model validation uses an external validation set for confirmation. This requires that the dataset under examination is sufficiently large, which is often not the case in the QSAR studies. If the dataset has a sufficiently large number of data points (i.e., molecules or compounds in QSAR) then, the dataset can be split up in a training and an external validation sets. This division can be made in several ways such as the D-optimal design [202], stratified validation samples etc. However, it is very important that both datasets should be representative and informative (i.e., they should span approximately similar ranges of the biological responses and the structural properties [203]).

Martens and Dardenne [187] demonstrated that for small datasets, independent validation test set are wasteful and still uncertain with sometimes over-optimistic estimates of future predictive error. The reason is that removing samples from an already limited set of available samples to an independent validation test set seriously biases the feature selection process and therefore reduces the predictive ability of the models, and while at the same time give uncertain, systematically over-optimistic assessment of the predictive ability of the models (because of "over-fitted" features).

In the absence of an external validation set, two reasonable ways of model validation can be performed by either a cross-validation that simulates how well the model predicts new data, or model re-estimation after randomization that estimates the chance probability to obtain a good fit with randomly shuffled response values [181].

9.4.2 Cross-validation and Bootstrapping without Replacement

Cross-validation is also called **internal validation**, since all the data points belong to same dataset that was also used in the training or building of a predictive model. In cross-validation, a dataset is divided into ***k*** subsets of approximately equal size. A **k**-fold-cross-validation procedure requires the fitting the model into dataset **k** times; at each time a subset is left out once, and only once, as a validation set and the remaining datasets are combined as a training set. If **k** is equal to the number of data points; then, it is called leave-one-out-cross-validation (LOO_{CV}). If **k** is less than the number of data points; then, it is called as leave-several-out-cross-validation (LSO_{CV}).

The LSO_{CV} leaves out a certain portion of the dataset; therefore, it creates a constant perturbation in the structure. On the other hand, the LOO_{CV} perturbs the data

structure by removing one data points at a time; therefore, causes an increasingly smaller perturbation with increasing the number of data points. Hence, $(1\text{-}Q^2)$ of LOO_{CV} approaches R^2 of LOO_{CV} in the limit [203]. Therefore, LOO_{CV} gives an over-optimistic result that is an underestimation of the true prediction error. On the other hand, LSO_{CV} (leaving approximately 7 molecules in the validation set) is recommend in order to get an unbiased estimation of the model quality [203].

In this dissertation, bootstrapping without replacement is employed for model validation instead of cross-validation. Bootstrapping without replacement is explained in section 8.7.1. The reason behind this choice is that there is a relationship between the bootstrapping without replacement and cross-validation methods. Since the cross-validation estimate is a heuristic estimate that depends on the division of data into folds, repeating cross-validation multiple times using different splits into folds and combining those estimates gives more accurate estimate. A complete cross-validation estimate is a combination of the estimates of all $\binom{n}{n/k}$ possibilities for choosing n/k instances out of n, where n is the number of data points and k is the number of folds [204]. The bootstrapping without replacement can be thought of as a better Monte Carlo estimate to the corresponding complete cross-validation.

9.4.3 Randomization test (Randomization of the Response Values)

A randomization test is based on repetitive random reassignments (or shuffling) of the order of the original values of the response (**Y**) variable in the training set. Then, a model is built on each of the modified (randomized) datasets and its prediction error is calculated. If in each case the modified (randomized) dataset gives significantly higher

Q^2 values than the original dataset; then, it can be inferred that the real QSAR model is relevant and predictive. In general, some hundreds of modeling of modified datasets are required, but randomization of the **Y** data a number of times (at least 10 times) gives a fairly good idea of the significance of the real QSAR model [203]. A target level of confidence for the randomization test can be set to 90, 95, 98, or 99 percent by comparing the resulting scores with the score of the original QSAR equation generated with the non-randomized (original) data. The higher the confidence level, the more randomization tests are to be performed. It is suggested that for a 90% confidence level, 9 trials are to be performed, 19 trials at 95%, 49 trials at 98%, and 99 trials at 99% [205]. If the training and prediction error of the random models are comparable to that of model with the original dataset, it can be concluded that either noise is modeled or the set of observations is not sufficient to support to build a predictive model.

A randomization test is very useful tool for the validation of the models with features selected by genetic algorithms in such situations [183]:

i) A dataset is very noisy (especially response variable).

ii) A dataset has a large number of descriptive features for a relatively small number of molecules.

If a GA is used for building a predictive model on these kinds of datasets, the GA may simply model the noise. A randomization test can be a useful validation tool to verify it.

9.4.4 Validation Strategy for GAFEAT-PLS

In order to properly validate the proposed GAFEAT-PLS feature selection method for small datasets, a combination of partial and full validation approaches is used. Partial

validation is applied to all models by using 100-bootstrap samples after the GAFEAT-PLS feature selection step. Full validation is applied to GAFEAT-PLS feature selection by employing a randomization test. The validation strategy consists of the following three steps:

i) The models constructed with features selected by GAFEAT-PLS are validated based on 100 bootstrap samples as explained in section 9.4.2. The main advantage of the bootstrapping without replacement over a LSO_{CV} is that the former method gives a better estimation.

ii) In the literature, some researchers have reported that although the wrapper feature selection methods have certain advantages, they select an optimal feature subset biased to a particular learning algorithm [6]. Since any learning algorithm is biased to some degree, selecting feature subsets tailored to a particular learning algorithm is equivalent to customizing the data to fit into that particular learning algorithm. The purpose of the feature selection is not only to build a good predictive model but also to explain and interpret to some degree how and why the model works. Therefore, especially in QSAR studies, a good feature subset, which is independent of any learning algorithm, will give more useful information that can be easily interpretable. Since GAFEAT-PLS is a wrapper feature selection method in which the PLS regression is used as a cost function, the quality of the selected feature subsets are also measured by another learning algorithm, in this case Support Vector Machines (SVM) regression, in addition to the PLS models.

iii) A randomization test is performed for the full GAFEAT-PLS feature selection method in order to verify that GAFEAT-PLS selects a good feature subset that model

information available in the dataset. To evaluate the statistical significance of a QSAR model constructed with features selected by a GAFEAT-PLS for an actual dataset, a standard hypothesis testing approach proposed by Zheng and Tropsha [100] is used. The null (**H_0**) and alternative (**H_A**) hypothesis are formulated as following:

$$H_0 : h = \mu$$

$$H_A : h < \mu \quad \text{or} \quad H_A : h > \mu$$

where **h** is the Q^2 or R^2 value for the model constructed with the selected features by GAFEAT-PLS from the original dataset and μ is the average value of the Q^2 or R^2 for the models constructed with the selected features by GAFEAT-PLS from random datasets. The null hypothesis states that the model for the original dataset is not significantly better than random models. If the training (R^2) and prediction (Q^2) error of the random models are comparable to that of model with the original dataset based on the hypothesis testing, it can be concluded that either GAFEAT-PLS simply models the noise (overfitting) or the set of observations is not sufficient to support to build a predictive model.

In standard hypothesis testing, the sampling distribution of the statistic is assumed to normal and the formula shown below is used for calculating the standardized test statistic (**Z**).

$$Z = \frac{h - \mu}{\sigma}$$

where s is the standard error of Q^2 or R^2 of the models constructed with the selected features by GAFEAT-PLS from random datasets. Value of the **Z** determines whether it is

appropriate to reject or not reject the null hypothesis based on a target level of confidence (e.g. 90, 95, 98, or 99 percent).

9.5 Computational Evaluation of GAFEAT-PLS

There are two QSAR datasets that have been used to evaluate the performance of GAFEAT-PLS: i) the Lombardo dataset originally with 62 compounds and 694 descriptive features (see section 7.2). ii) the HIV_{rt} dataset with 64 compounds and 620 descriptive features (see section 7.3).

The first step is to use objective feature selection (e.g. removing non-changing, highly inter-correlated (*cousin features*), and 4-sigma outlier features) to remove features that contain redundant, minimal information or distorted information. The StripMiner™ program [4], which is a general purpose data preprocessing and modeling program for the scientific data mining of large datasets, is used to perform objective feature selection. The StripMiner™ program removes cousin (highly inter-correlated) features in the following way: if two features are correlated with each other above a pre-specified correlation threshold, StripMiner™ removes the feature that is less correlated with the response variable. In general, this correlation threshold is set to 95%. StripMiner™ is also used to remove the 4σ outliers. Since the QSAR datasets studied in this dissertation are characterized by a large number of descriptive features for a relatively small number of molecules, only the outlier features rather than molecules with outlier features are removed.

9.5.1 The Lombardo Dataset

The Lombardo dataset was pre-processed with StripMiner™ in order to remove non-changing, highly inter-correlated (%95 and above) features, and features with 4σ outlier. After pre-processing, 309 descriptive features were retained in the Lombardo dataset.

Since GAFEAT-PLS conducts a search for a good feature subset using the PLS regression itself as part of the fitness function, the accuracy estimation of the PLS model may be overly optimistic (overfits to the PLS regression model). Therefore, another learning algorithm, namely the SVM regression, is used for constructing a predictive model and its accuracy estimation (validation error) is compared with that of the PLS regression model. In a summary, it is desired to find a good feature subset, which give consistent results for both the PLS and SVM regression models.

First of all, the StripMiner™ [4] program is employed to construct predictive models (the PLS and SVM regression) of the Lombardo dataset with 309 original features. The predictive abilities of the PLS and SVM regression models are obtained based on 100-bootstrapping samples by leaving out 6 molecules in each validation set. The PLS regression models described in this dissertation used four latent variables for all calculations. However, the SVM regression in the StripMiner™ [4] automatically computes the parameters of the SVM regression model (the kernel parameter σ, the trade-off constant C, and the value of ε) for each bootstrapping sample by using a pattern search algorithm [146, 149]. For this reason, validations of the SVM regression models are computationally expensive. The results of the 100 bootstrap validation of the PLS and SVM regression models with all features are presented in the Table 9.3.

Table 9.3 100-bootstrap Validation of Full Lombardo Dataset

Learning Models	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
PLS	0.9228	0.9228	0.2853	0.2858
SVM	0.9846	0.9822	0.3098	0.3105

GAFEAT-PLS is applied on the Lombardo dataset to select a good feature subset with a minimum number of features. Since the number of selected features is predetermined, several of GAFEAT-PLS runs with different numbers of features are performed in order to determine the most informative feature subset with the minimum number of descriptive features. The parameters of GAFEAT-PLS used for the Lombardo dataset are listed in the Table 9.4.

Table 9.4 Parameters of GAFEAT-PLS for the Lombardo Dataset

Population Size	Crossover Probability	Mutation Probability	Maximum # of Generation	Number of Latent Variables
100	0.90	0.02	1000	4
Number of Bootstraps in the Fitness Evaluation		Number of Molecules in the Training Set		Number of Molecules in the Validation Set
20		56		6

GAFEAT-PLS is executed 5 times to select 10, 20, 30, 40, and 50 features from the Lombardo dataset with 309 features. Each selected feature subset is modeled by the PLS and SVM regression, and are further validated with 100-boostrap samples by leaving 56 molecules in training sets and 6 molecules in the validation sets. The feature subset validation results of the PLS and SVM regression models are presented in the Tables 9.5 and 9.6, respectively.

Table 9.5 100-bootstrap Validation Results of PLS Models of the Feature Subset Selected by GAFEAT-PLS from the Lombardo Dataset

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.9122	0.9122	0.1710	0.1729
20	0.9719	0.9719	0.0809	0.0822
30	0.9812	0.9812	0.0669	0.0677
40	0.9801	0.9801	0.0749	0.0755
50	0.9828	0.9828	0.0831	0.0871

Table 9.6 100-bootstrap Validation Results of SVM Models of the Feature Subset Selected by GAFEAT-PLS from the Lombardo Dataset

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.9320	0.9309	0.1922	0.1951
20	0.9731	0.9705	0.0962	0.0966
30	0.9810	0.9795	0.0719	0.0733
40	0.9821	0.9802	0.0950	0.0955
50	0.9824	0.9795	0.0981	0.1018

Figure 9.7 shows the number of features selected by GAFEAT-PLS versus 100-bootstrap validation errors of the PLS and SVM regression models of the corresponding selected feature subsets in terms of Q^2 statistics. The PLS and SVM regression models with the selected features are significantly more predictive than those models containing all the features. It can be seen that the subset with 30 features is the best subset since its validation error is the lowest one (also *relatively low*) when compared with other feature subsets. '*Relatively low validation error*' means that a feature subset gives the lowest error as the average of the validation errors of the SVM and PLS regression models. It is

hypothesized in this dissertation that a good feature subset should give similar results for the PLS and SVM regression models.

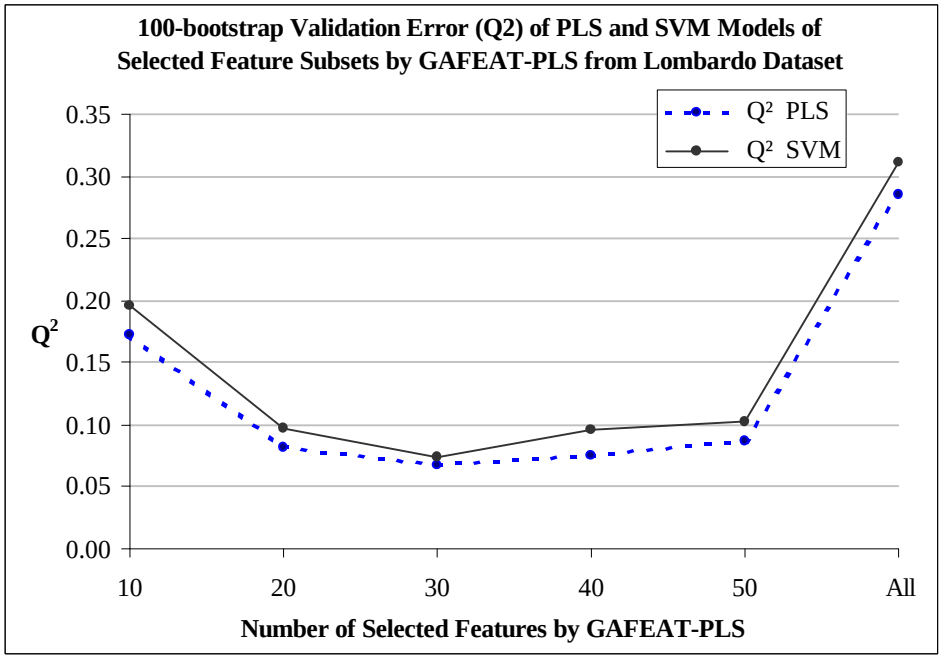


Figure 9.7 100-bootstrap Validation Errors (Q^2) of the PLS and SVM Regression Models of Selected Feature Subsets by GAFEAT-PLS from the Lombardo Dataset

9.5.2 The HIV_{rt} Dataset

HIV_{rt} dataset has been pre-processed by StripMiner™ [4] to remove non-changing, highly correlated (%95 and above) and 4σ outlier features. After pre-processing, 230 descriptive features left in the HIV_{rt} dataset. The StripMiner™ program is employed to construct predictive models (the PLS and SVM regressions) of the HIV_{rt} dataset with 230 original features. The obtained PLS and SVM models are validated

based on 100-bootstrapping samples by leaving 58 molecules in training set and 6 molecules in the validation set. These validation results are presented in the Table 9.7.

Table 9.7 100-bootstrap Validation of Full HIV_{rt} Dataset

Learning Models	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
PLS	0.8129	0.8129	0.3354	0.3368
SVM	0.9276	0.9159	0.3690	0.3718

GAFEAT-PLS is applied to select a good feature subset with the minimum number of features for the HIV_{rt} dataset. The parameters of GAFEAT-PLS used for the HIV_{rt} dataset is presented in the Table 9.8. Since the number of feature to be selected is predetermined, a several of GAFEAT-PLS runs are performed to obtain the most informative and predictive feature subset with the minimum number of features.

Table 9.8 Parameters of GAFEAT-PLS for the HIV_{rt} Dataset

Population Size	Crossover Probability	Mutation Probability	Maximum # of Generation	Number of Latent Variables
100	0.90	0.02	1000	4
Number of Bootstraps in the Fitness Evaluation		Number of Molecules in the Training Set		Number of Molecules in the Validation Set
20		58		6

GAFEAT-PLS executed to select 10, 20, 30, 40, and 50 features from the HIV_{rt} dataset with 230 features. Each selected feature subset is modeled by the PLS and SVM regression techniques, and these models are further validated with 100-boostrapping

samples by leaving 58 molecules in training set and 6 molecules in the validation set. The feature subset validation results for the PLS and SVM regression models are presented in the Tables 9.9 and 9.10, respectively.

Table 9.9 100-bootstrap HIV_{rt} Feature Subset Validation with PLS Regression

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.8712	0.8712	0.1843	0.1858
20	0.9142	0.9142	0.1592	0.1593
30	0.9362	0.9362	0.1840	0.1851
40	0.9379	0.9379	0.1966	0.1969
50	0.9299	0.9299	0.1912	0.1915

Table 9.10 100-bootstrap HIV_{rt} Feature Subset Validation with SVM Regression

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.8807	0.8793	0.2194	0.2218
20	0.9218	0.9182	0.1865	0.1872
30	0.9753	0.9749	0.1456	0.1470
40	0.9796	0.9788	0.1602	0.1618
50	0.9703	0.9683	0.2220	0.2237

Figure 9.8 shows the plot of the number of selected features by GAFEAT-PLS versus the 100-bootstrap validation errors of the PLS and SVM models for the corresponding selected feature subsets in terms of Q^2 statistics. It is seen from Figure 9.8 that the feature subset with 30 features is the best subset since whose validation error is relatively lower when compared to other feature subsets.

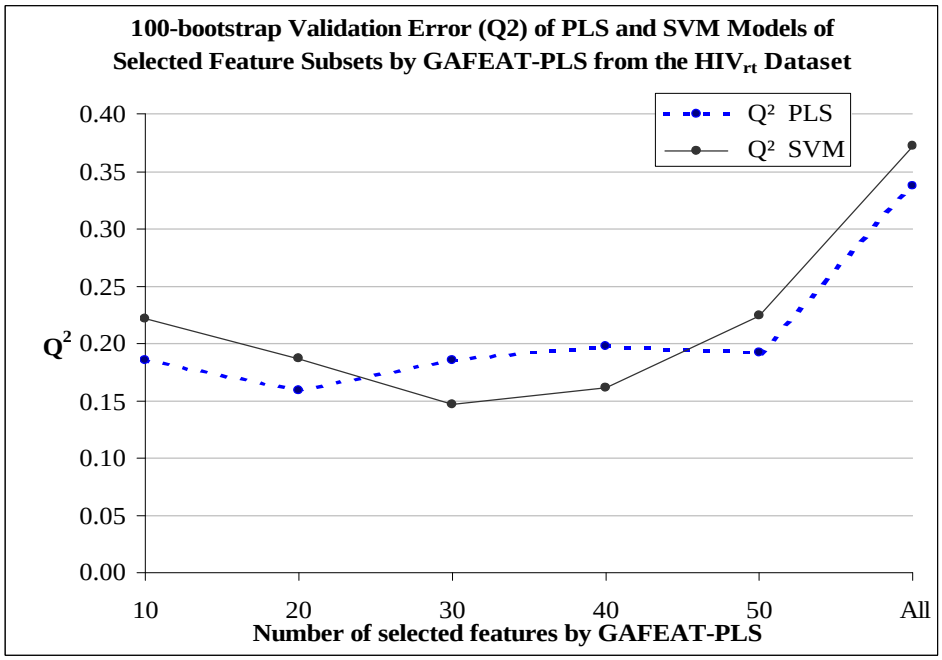


Figure 9.8 100-bootstrap Validation Errors (Q^2) of the PLS and SVM Regression Models of Selected Feature Subsets by GAFEAT-PLS from the HIV_{rt} Dataset

9.6 Computational Validation of GAFEAT-PLS with a Randomization Test

As explained in section 9.4.4, the validation (robustness) of GAFEAT-PLS method is examined with a standard hypothesis testing where the results of a real dataset is compared to those of randomized datasets. The randomization test validation of GAFEAT-PLS is performed on the Lombardo and HIV_{rt} datasets. Full validation result of GAFEAT-PLS based on 10 fold cross-validation and leave-one-out cross-validation on the Lombardo and HIV_{rt} datasets are presented in the Appendix C. These full validation results are presented in the appendix, since full validation approach for feature selection in small datasets is not conclusive as explained in section 9.4.

9.6.1 The Lombardo Dataset

Ten randomization trails were performed in which the biological activity data was randomized with respect to the dataset. A full GAFEAT-PLS feature selection method is performed for each randomized dataset. Randomization of data, GAFEAT-PLS feature selection, and subsequent model evaluation are performed to assess the statistical validity of QSAR models and the overall GAFEAT-PLS feature selection method. Each selected feature subset is modeled using a PLS regression with 4 latent variables and the predictive quality of the resulting model is measured based on 100 bootstrap samples leaving 56 molecules in the training sets and 6 molecules in the validation sets. Figures 9.9 and 9.10 show the results of randomization test of GAFEAT-PLS feature selection method on the Lombardo dataset in terms of the Q^2 and R^2 statistics, respectively.

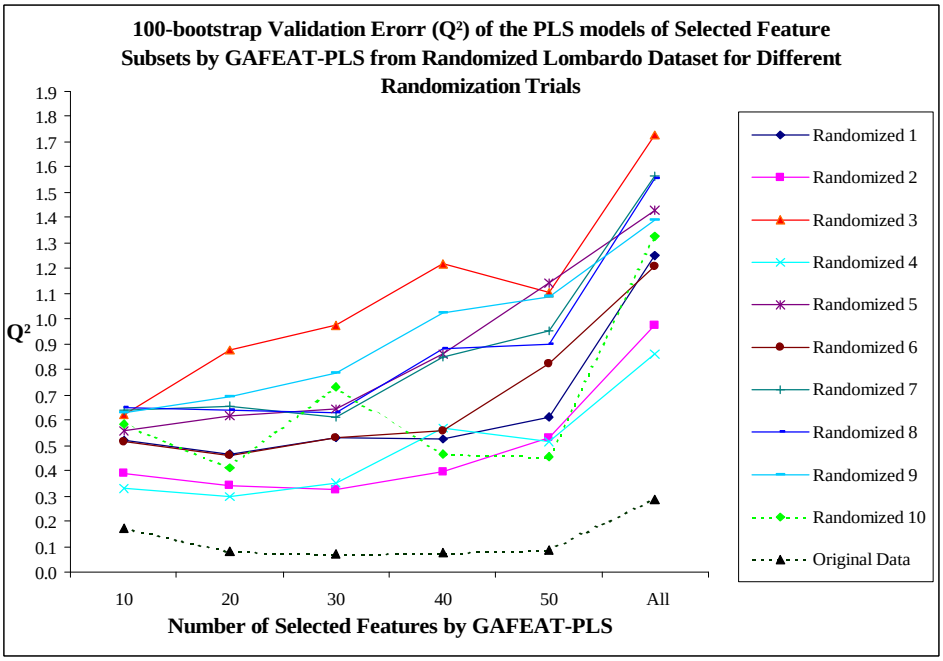


Figure 9.9 100-bootstrap Validation Errors (Q^2) of the PLS Models of Feature Subsets Selected by GAFEAT-PLS from Randomized Lombardo Dataset for Different Randomization Trials

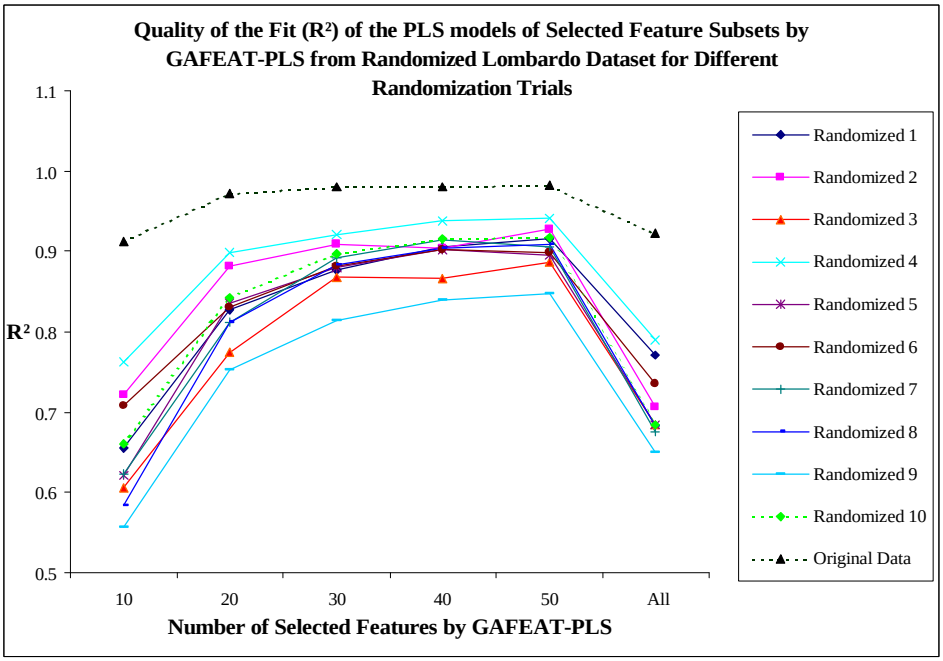


Figure 9.10 Quality of the Fit (R^2) of the PLS Models of Selected Feature Subsets by GAFEAT-PLS from the Randomized Lombardo Dataset for Different Randomization Trials

It appears from Figures 9.9 and 9.10 that in each randomization test the randomized data give much higher Q^2 and much lower R^2 values than those of the original data. As explained in section 9.4.4, the validation (robustness) of GAFEAT-PLS method is carried out by standard hypothesis testing where the results of a real dataset is compared to those of randomized datasets. The hypothesis testing results based on R^2 and Q^2 are presented in Tables 9.11 and 9.12, respectively. Z score and P value represent the test statistics and its probability that is the smallest probability leading to rejection of the null hypothesis, respectively.

Table 9.11 Standard One-Tail Hypothesis Testing of GAFEAT-PLS on the Lombardo Dataset Based on R^2

Number of Selected Features	Original Data	Mean of the Randomization Trials	Std. Dev. of the Randomization Trials	Z score	P value
10	0.9122	0.6495	0.0650	4.0398	0.0000
20	0.9719	0.8269	0.0435	3.3340	0.0004
30	0.9812	0.8821	0.0290	3.4196	0.0003
40	0.9801	0.8992	0.0273	2.9618	0.0015
50	0.9828	0.9047	0.0253	3.0881	0.0010
All	0.9228	0.7062	0.0446	4.8570	0.0000

Table 9.12 Standard One-Tail Hypothesis Testing of GAFEAT-PLS on the Lombardo Dataset Based on Q^2

Number of Selected Features	Original Data	Mean of the Randomization Trials	Std. Dev. of the Randomization Trials	Z score	P value
10	0.1729	0.5435	0.1089	-3.4014	0.0003
20	0.0822	0.5452	0.1807	-2.5626	0.0052
30	0.0677	0.6109	0.1944	-2.7947	0.0026
40	0.0755	0.7354	0.2702	-2.4424	0.0073
50	0.0871	0.8122	0.2653	-2.7328	0.0031
All	0.2858	1.3286	0.2677	-3.8956	0.0000

Based on P values, it can be concluded with a 90 percent of confidence that GAFEAT-PLS models on original datasets give consistently higher R^2 and lower Q^2 than those of GAFEAT-PLS models obtained on the ten different randomized datasets.

9.6.2 The HIV_{rt} Dataset

Ten randomization trails were performed in which the biological activity data was randomized with respect to the dataset. A full GAFEAT-PLS feature selection is performed for each randomized HIV_{rt} dataset. Randomization of data, GAFEAT-PLS feature selection and subsequent model evaluations are performed to assess the statistical validity of the QSAR model and GAFEAT-PLS feature selection method. Each selected

feature subset is modeled using the PLS algorithm with 4 latent variables and the predictive quality of the resulting model is measured based on 100 bootstrap samples leaving 6 molecules in the validation set. Figures 9.11 and 9.12 show the results of randomization test of GAFEAT-PLS feature selection method on the HIV_{rt} dataset.

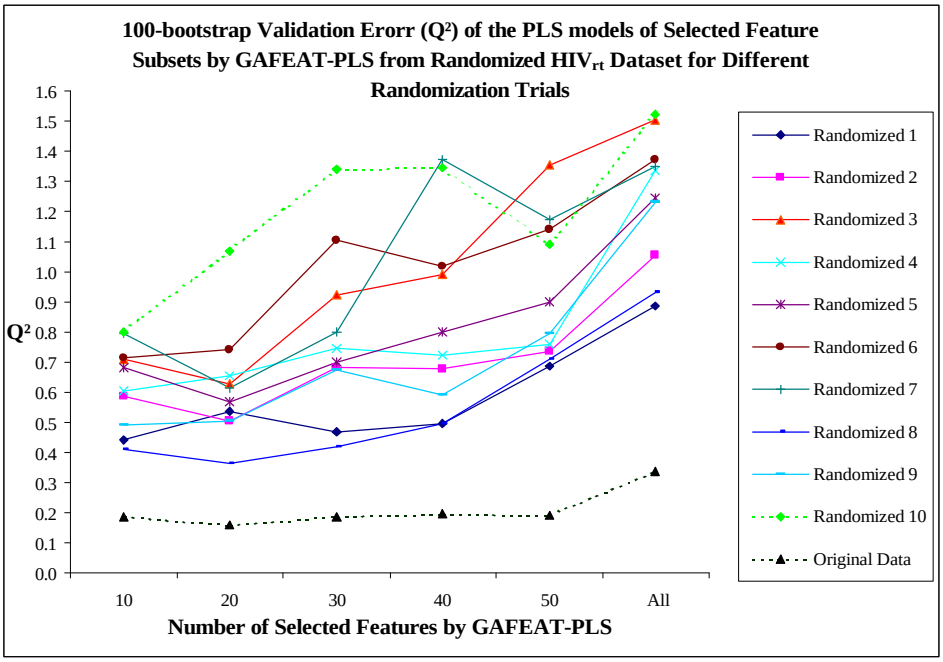


Figure 9.11 100-bootstrap Validation Errors (Q^2) of the PLS Models of Selected Feature Subsets by GAFEAT-PLS from the Randomized HIV_{rt} Dataset for Different Randomization Trials

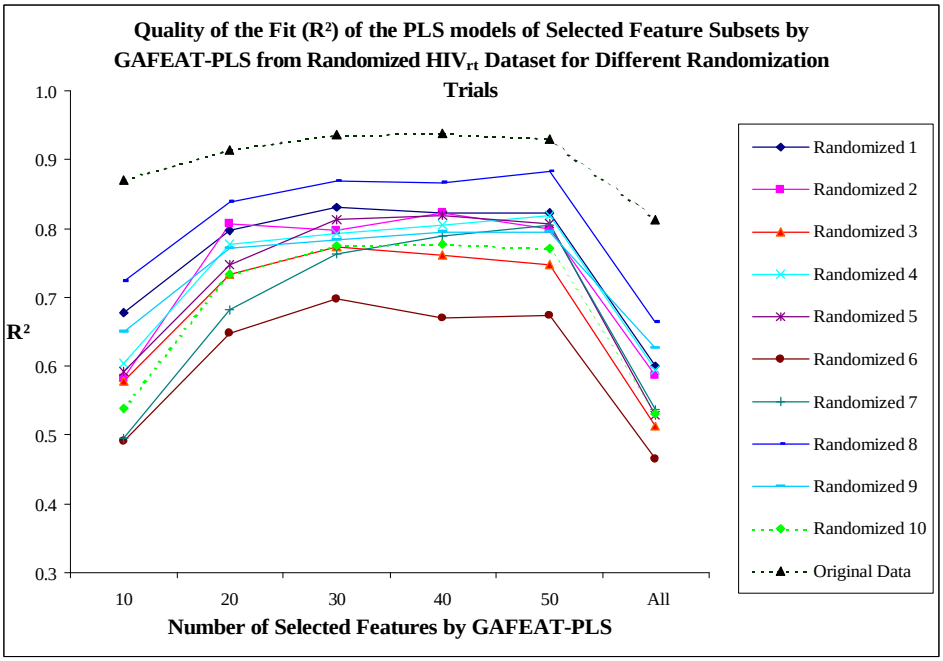


Figure 9.12 Quality of the Fit (R^2) of the PLS Models of Selected Feature Subsets by GAFEAT-PLS from the Randomized HIV_{rt} Dataset for Different Randomization Tests

It appears from the Figures 9.11 and 9.12 that in each randomization test the randomized data give much higher Q^2 and much lower R^2 values than those of the original data. The hypothesis testing results based on R^2 and Q^2 are presented in Tables 9.13 and 9.14, respectively. Z score and P value represent the test statistics and its probability that is the smallest probability leading to rejection of the null hypothesis, respectively.

Table 9.13 Standard One-Tail Hypothesis Testing of GAFEAT-PLS on the HIV_{rt} Dataset Based on R^2

Number of Selected features	Original Data	Mean of the Randomization Trials	Std. Dev. of the Randomization Trials	Z score	P value
10	0.8712	0.5929	0.0754	3.6903	0.0001
20	0.9142	0.7535	0.0579	2.7733	0.0028
30	0.9362	0.7895	0.0453	3.2417	0.0006
40	0.9379	0.7929	0.0524	2.7699	0.0028
50	0.9299	0.7923	0.0546	2.5203	0.0059
All	0.8129	0.5648	0.0596	4.1616	0.0000

Table 9.14 Standard One-Tail Hypothesis Testing of GAFEAT-PLS on the HIV_{rt} Dataset Based on Q^2

Number of Selected features	Original Data	Mean of the Randomization Trials	Std. Dev. of the Randomization Trials	Z score	P value
10	0.1858	0.6228	0.1408	-3.1033	0.0010
20	0.1593	0.6179	0.1878	-2.4419	0.0073
30	0.1851	0.7852	0.2780	-2.1587	0.0154
40	0.1969	0.8510	0.3219	-2.0319	0.0211
50	0.1915	0.9347	0.2366	-3.1419	0.0008
All	0.3368	1.2436	0.2219	-4.0862	0.0000

Based on the P values of the test statistics, it can be concluded with a 90 percent of confidence that GAFEAT-PLS models on original datasets give consistently higher R^2 and lower Q^2 than those of GAFEAT-PLS models obtained on the ten different randomized datasets.

9.7 Computational Evaluation of the Implicit Nonlinear GAFEAT-PLS

It is worth mentioning that PLS model is a linear while the SVM model is non-linear. If the relationship between the predictor data (X) and response data (Y) is linear, then it is expected that nonlinear model (SVM) will converge to a linear model. However, almost all of the real datasets (i.e., such as QSAR datasets used in this dissertation) exhibit moderate nonlinear characteristics. Since PLS is a linear model, GAFEAT-PLS selects feature subsets based on a linear criterion, even though the dataset has some nonlinear characteristics. These feature subsets selected by GAFEAT-PLS may also contain some nonlinear characteristics depending on the dataset under scrutiny. These nonlinear characteristics cannot be taken into consideration by GAFEAT-PLS, since the PLS algorithm only models linear relationships. Depending on the existing nonlinearities in the selected feature subsets, the SVM regression models sometimes may perform better than PLS. It is expected that if PLS regression could also model for nonlinearities,

its performance will be similar to that of the SVM model. This issue is addressed by applying the INLR method proposed in [192] on the Lombardo and HIV_{rt} datasets. The INLR method has been explained in the section 9.1.5.

9.7.1 The Lombardo dataset

Lombardo dataset originally had 62 compounds and 694 descriptive features (see section 7.2). First, the squares of the features were added into the Lombardo dataset and the total number of features at the onset of the analysis became 1388. The expanded Lombardo dataset has been pre-processed with StripMiner™ in order to remove non-changing, highly correlated (%95 and above) and the 4σ outlier features. After pre-processing, 442 descriptive features remained in the expanded Lombardo dataset (241 original features and 241 squared features).

The StripMiner™ [4] program is then employed to construct PLS and SVM predictive regression models of the expanded Lombardo dataset consisting of 442 features. The obtained PLS and SVM regression models are validated based on 100-bootstrapping samples by leaving out 6 molecules in the validation set each time. These validation results are presented in the Table 9.15.

Table 9.15 100-bootstrap Validation of the Expanded Lombardo Full Dataset

Learning Models	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
PLS	0.9315	0.9315	0.2595	0.2595
SVM	0.9482	0.9404	0.2798	0.2802

GAFEAT-PLS is applied to the expanded Lombardo dataset to select a good subset with the minimum number of descriptive features. The same parameter set listed in the Table 9.4 is used for GAFEAT-PLS. GAFEAT-PLS is run to select 10, 20, 30, 40, and 50 features from the expanded Lombardo dataset with 442 features. Each selected feature subset is modeled by the PLS and SVM regression in the StripMiner™ [4] and are further validated with 100-boostrapping samples by leaving 56 molecules in training set and 6 molecules in the validation set. These feature subset validation results for the PLS and SVM regression models are presented in the Tables 9.16 and 9.17, respectively.

Table 9.16 100-bootstrap the Expanded Lombardo Feature Subset Validation with PLS Regression

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.9122	0.9122	0.1710	0.1729
20	0.9794	0.9794	0.0511	0.0513
30	0.9840	0.9840	0.0570	0.0571
40	0.9877	0.9877	0.0559	0.0572
50	0.9890	0.9890	0.0544	0.0567

Table 9.17 100-bootstrap the Expanded Lombardo Feature Subset Validation with SVM Regression

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.9320	0.9309	0.1922	0.1951
20	0.9859	0.9855	0.0565	0.0569
30	0.9913	0.9911	0.0591	0.0593
40	0.9888	0.9875	0.0532	0.0548
50	0.9934	0.9928	0.0504	0.0521

Figure 9.13 shows the number of selected features from the expanded Lombardo dataset by GAFEAT-PLS versus the 100-bootstrap validation errors of the PLS and SVM regression models of the corresponding selected feature subsets in terms of Q^2 statistics. The PLS and SVM regression models with the selected features are significantly more predictive than those models with all features. Here, although feature subsets with 20, 30, 40, and 50 features give almost similar validation error, it is seen that the feature subset with 20 features is the best subset since its validation error is relatively lower and contains less number of features when compared with other feature subsets.

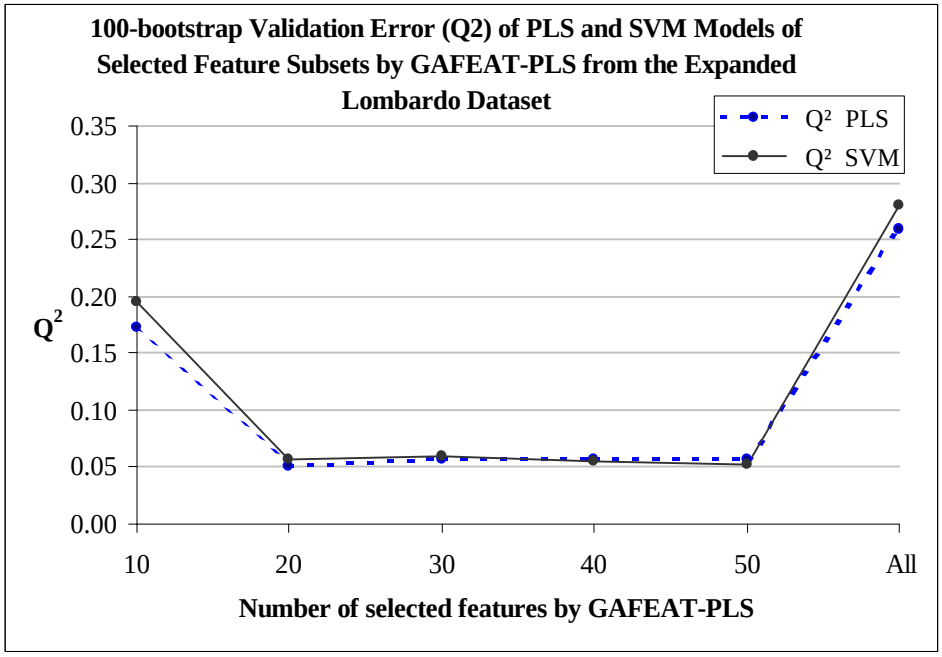


Figure 9.13 100-bootstrap Validation Errors of the PLS and the SVM Regression Models of Selected Feature Subsets by GAFEAT-PLS from the Expanded Lombardo Dataset

9.7.2 The HIV_{rt} dataset

HIV_{rt} dataset originally had 64 compounds and 620 descriptive features (see section 7.3). First, the squares of the features were added into the HIV_{rt} dataset and the total number of features became 1240. The expanded HIV_{rt} dataset has been pre-processed in order to remove non-changing, the highly correlated (%95 and above) and the 4-sigma outlier features. After pre-processing, 369 descriptive features remained in the HIV_{rt} dataset (197 original features and 172 squared features).

The StripMiner™ [4] program is employed to construct PLS and SVM predictive regression models of the expanded HIV_{rt} dataset containing 369 features. The obtained PLS and SVM models are validated based on 100-bootstrapping samples by leaving out 6 molecules in the validation set each time. These validation results are presented in the Table 9.18.

Table 9.18 100-bootstrap Validation of the Expanded HIV_{rt} Full Dataset

Learning Models	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
PLS	0.8103	0.8103	0.3326	0.3349
SVM	0.9319	0.9215	0.3546	0.3546

GAFEAT-PLS is applied to the expanded HIV_{rt} dataset to select a good feature subset with the minimum number. The same parameter set listed in the Table 9.4 is used for GAFEAT-PLS. GAFEAT-PLS is used to select 10, 20, 30, 40, and 50 features from the expanded HIV_{rt} dataset with 369 features. Each selected feature subset is modeled by the PLS and SVM regression methods, and these models are further validated with 100-

boostrapping samples. These feature subset validation results for the PLS and the SVM regressions are presented in the Tables 9.19 and 9.20, respectively.

Table 9.19 100-bootstrap the Expanded HIV_{rt} Feature Subset Validation with PLS

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.8805	0.8805	0.1581	0.1610
20	0.9149	0.9149	0.1337	0.1347
30	0.9366	0.9366	0.1073	0.1089
40	0.9272	0.9272	0.1401	0.1437
50	0.9336	0.9336	0.1157	0.1204

Table 9.20 100-bootstrap the Expanded HIV_{rt} Feature Subset Validation with SVM

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.8976	0.8976	0.1937	0.1957
20	0.9252	0.9200	0.1652	0.1690
30	0.9554	0.9521	0.1336	0.1371
40	0.9416	0.9366	0.2072	0.2128
50	0.9464	0.9406	0.1855	0.1901

Figure 9.14 shows the number of selected features from the expanded HIV_{rt} dataset by GAFEAT-PLS versus the 100-bootstrap validation errors of the PLS and SVM regression models of the corresponding selected feature subsets in terms of Q^2 statistics. The PLS and SVM regression models with the selected features are significantly more predictive than those models with all features. Here, it is seen that the feature subset with 30 features is the best subset since whose validation error is the lowest (also relatively lowest) when compared with other feature subsets.

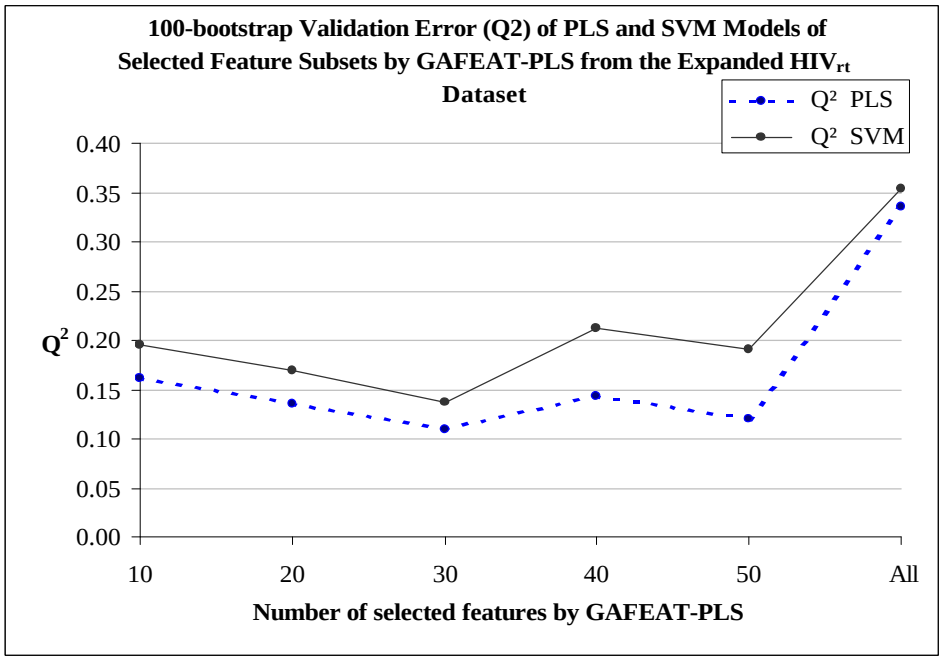


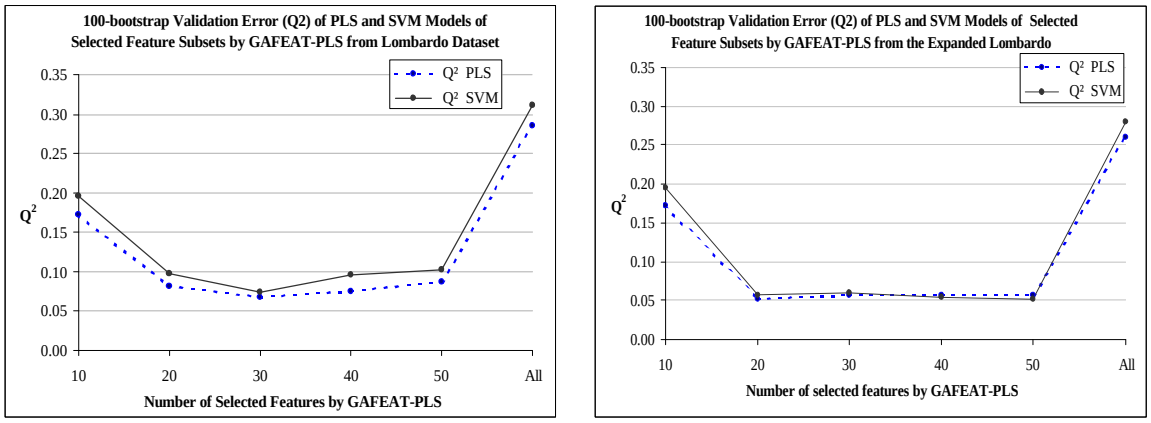
Figure 9.14 100-bootstrap Validation Errors of the PLS and the SVM Regression Models of Selected Feature Subsets by GAFEAT-PLS from the Expanded HIV_{rt} Dataset

9.7.3 Comparisons of GAFEAT-PLS and Implicit Nonlinear GAFEAT-PLS

Like multiple regression models, PLS models give a linear relationship between X and Y data. However, almost all of the real datasets (i.e. such as QSAR datasets used in this dissertation) exhibit nonlinear characteristics in some degree. A wide degree of non-linearity (from mild to severe) may exist between sets of variables, and the mildest non-linearities can be modeled by quadratic polynomials [195]. Berglund et al. point out that if a dataset has mild non-linear characteristics, the INLR method works well, and if non-linearities are more severe than quadratically polynomial, they cannot be modeled by the INLR method [195].

GAFEAT-PLS feature selection method integrated with the INLR method has been applied to the Lombardo and HIV_{rt} datasets in sections 9.7.1 and 9.7.2, respectively.

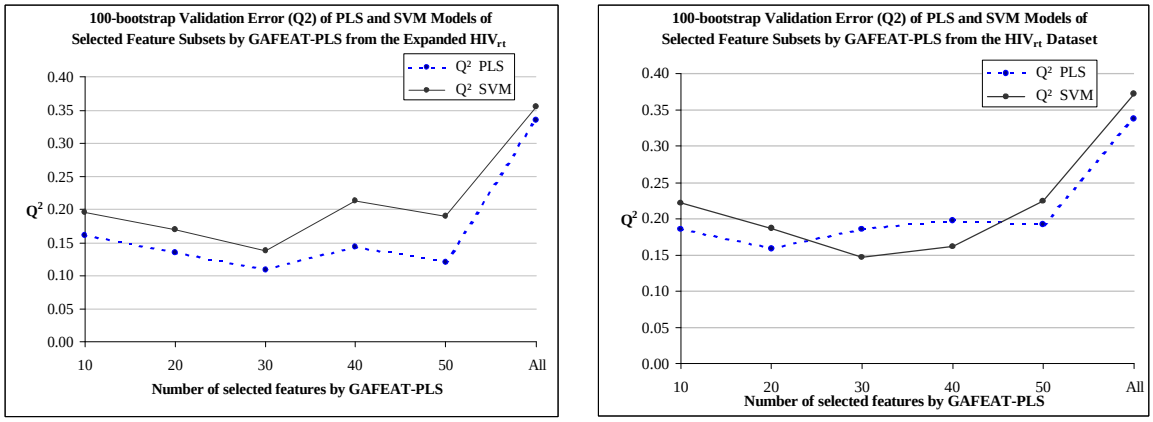
The comparisons of GAFEAT-PLS feature selection method without and with the INLR method are presented for the Lombardo and HIV_{rt} datasets in Figures 9.15 and 9.16, respectively. It is clearly seen from these computational results that the INLR models perform better than the PLS models on the datasets with all features (without feature selection). As we previously pointed out that if the PLS regression is able to model the nonlinearities, and then its performance will be similar to that of the SVM model. This can be apparently seen from the Lombardo dataset. The performances of the PLS and SVM regression models with feature subsets selected by GAFEAT-PLS from the expanded Lombardo dataset give very similar prediction error in terms of Q^2 statistics calculated based on 100 bootstrap sample (see Figure 9.15-A). These results confirm that the Lombardo dataset has mild non-linear characteristics, which can be modeled by quadratic polynomials.



(A) **(B)**

Figure 9.15 Lombardo Dataset **(A)** GAFEAT-PLS **(B)** GAFEAT-PLS with the INLR Method

The results of the HIV_{rt} dataset show that the HIV_{rt} dataset has more severe non-linearities than quadratically polynomial, and they cannot be modeled by the INLR method totally (see Figure 9.16). As mentioned in the section 7.3 the HIV_{rt} dataset is a more non-linear dataset that contains 64 molecules representing five structural classes of reverse transcriptase inhibitors. Although the INLR method helps improving the predictive quality of the PLS models, there are more non-linearities remained as a noise. This can be seen from Figure 9.16 that there are some discrepancies between performances of the PLS and SVM regression models.



(A) **(B)**

Figure 9.16 HIV_{rt} Dataset **(A)** GAFEAT-PLS **(B)** GAFEAT-PLS with the INLR Method

Computational results show that GAFEAT-PLS is able to selected good feature subsets that result more predictive models than that of full dataset. The integration of the INLR method with GAFEAT-PLS further improves the quality of the feature subsets by taking into consideration mild non-linerities in the datasets.

9.8 Convergence of GAFEAT-PLS

The effectiveness of GAFEAT-PLS is demonstrated on real QSAR (HIV_{rt} and Lombardo) datasets. The feature subsets selected by GAFEAT-PLS apparently lead to more predictive models with fewer features than the models that incorporate all features. Although GAFEAT-PLS efficiently and effectively explores a large feature space and finds good predictive feature subsets based on a defined criterion, it cannot be known whether the features subsets are optimal (the best) for the criterion used. The only way to be sure of the convergence of GAFEAT-PLS would be to enumerate and evaluate all the possible PLS models. Since the whole search space in a feature subset selection problem with T features is $2^{\boldsymbol{T}}$, an exhaustive search is not applicable to the studied QSAR datasets that have over 200 features. Even though the number of features to be selected is predefined in GAFEAT-PLS, the optimal feature subset of size $\boldsymbol{N}$ chosen from a total of $\boldsymbol{T}$ features can be found by enumerating and testing all possibilities, which requires $\binom{T}{N} = \frac{T!}{N!(T-N)!}$ PLS models to be evaluated. This is also prohibitively expensive in computing time when T becomes larger. For instance, if T = 100, and N = 20, the number of models evaluated will be 5.35983E+20.

Instead of enumerating all possible subsets for the QSAR datasets, the convergence of GAFEAT-PLS is tested by using the subsets of features selected by GAFEAT-PLS. Since GAFEAT-PLS uses a cost function based on R^2, which is similar to that of the best subsets regression method, the best subset regression method can be used to test the convergence of GAFEAT-PLS.

The best subsets regression is a way to select a group of "best subsets" for further analysis by selecting the smallest subset that fulfills certain statistical criteria (e.g.

Mallow's Cp statistic, adj.R^2, RMSE). The subset model may actually estimate the regression coefficients and predict future responses with smaller variance than the full model using all predictors [206]. In the best subsets regression method, all possible regressions are performed, and the highest R^2 at each model size is recorded. It is then possible to explore the trade-offs between model fit and size by plotting this information. However, R^2 always increases with the size of the subset [126]. For instance, the best regression model with 5 features will always have a higher R^2 than that of the best model with 4 features. Therefore, R^2 is a useful criterion when comparing linear regression models of the same size. In this situation, choosing the linear regression model with the highest R^2 is equivalent to choosing the model with the smallest sum of square of error (SSE). The adjusted R^2 (adj.R^2) is more appropriate when comparing linear regression models with different number of features since adj.R^2 adjusts (penalizes) for the number of predictors (features) in the model [126]. The formula for adj.R^2 is defined by

$$\text{adj.}R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 / (n-p)}{\sum_{i=1}^{n} (y_i - \bar{y})^2 / (n-1)},$$

where **n** is the number of data points and **p** is the number of predictors in the model. In this case, choosing the linear regression model with the highest adj.R^2 is equivalent to choosing the model with the smallest mean square error (MSE). The best subset regression method becomes intractable due to the exponential growth of the number of possible subsets when the number of features becomes large. Minitab, a statistical program, can perform the best subsets regression for small size of regression problems (up to 20 features). The S-PLUS statistical program performs All-Subset Regressions by using leaps and bounds method [207] up to 30 features.

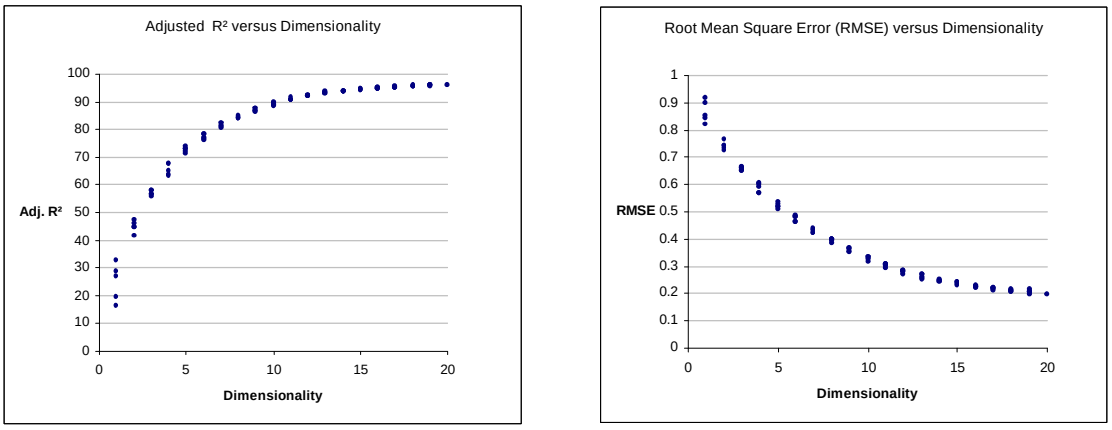
It can be hypothesized that a feature subset with size **N** selected by GAFEAT-PLS from a dataset with **T** features is a good feature subset if and only if it has no feature subsets performing better than itself and the **N** is less than or equal to the size of the best feature subset for the dataset with **T** features.

Using the adj.R^2 as a feature selection criterion, the best subsets regression of Minitab release 12.2 and all-subsets regression of S-PLUS 4.5 are used to evaluate feature subsets selected by GAFEAT-PLS from the Lombardo and HIV_{rt} datasets up to 30 features. It has been found that no subsets of those subsets with up to 20 features selected by GAFEAT-PLS performed better than the original feature subsets based on the adj.R2. This results are consistent with GAFEAT-PLS feature selection analysis applied on the HIV_{rt} and Lombardo datasets since the size of the optimal feature subsets for both datasets are greater than 20.

As an example to this experiment, the results of a feature subset with 20 features selected by GAFEAT-PLS from the Lombardo dataset are presented in Figure 9.17 in which the adj.R^2 and RMSE of the best 5 models of each dimensionality versus dimensionality are plotted. It can be clearly seen from Figure 9.17 that the best dimensionality is 20, which means that none of the 1,048,575 evaluated feature subsets performed better than the original subset selected by GAFEAT-PLS.

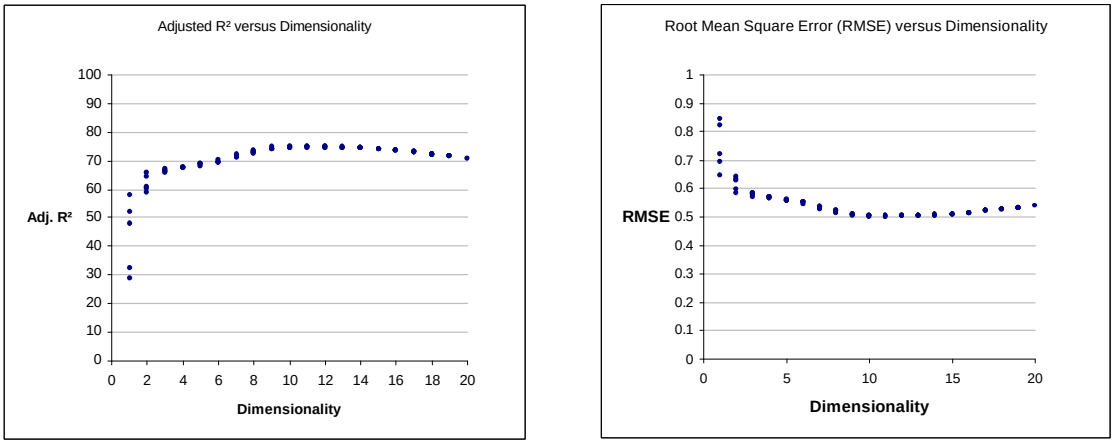
If GAFEAT-PLS had not been able to convergence to a good feature subset there would have been a dimensionality less than 20. In order to demonstrate this case, 20 features are randomly selected from the Lombardo dataset and the best subset regression is applied to this feature subset. The results of the best subset regression are presented in

the Figure 9.18. It is apparently seen that the best dimensionality for the random feature subset is 10, which is less 20 features.



(A) **(B)**

Figure 9.17 The Best Subset Regression Results of the Subset with 20 Features Selected by GAFEAT-PLS from the Lombardo Dataset **(A)** Adj.R^2 versus Dimensionality **(B)** RMSE versus Dimensionality



(A) **(B)**

Figure 9.18 The Best Subset Regression Results of the Subset with 20 Features Randomly Selected from the Lombardo Dataset **(A)** Adj.R^2 versus Dimensionality **(B)** RMSE versus Dimensionality

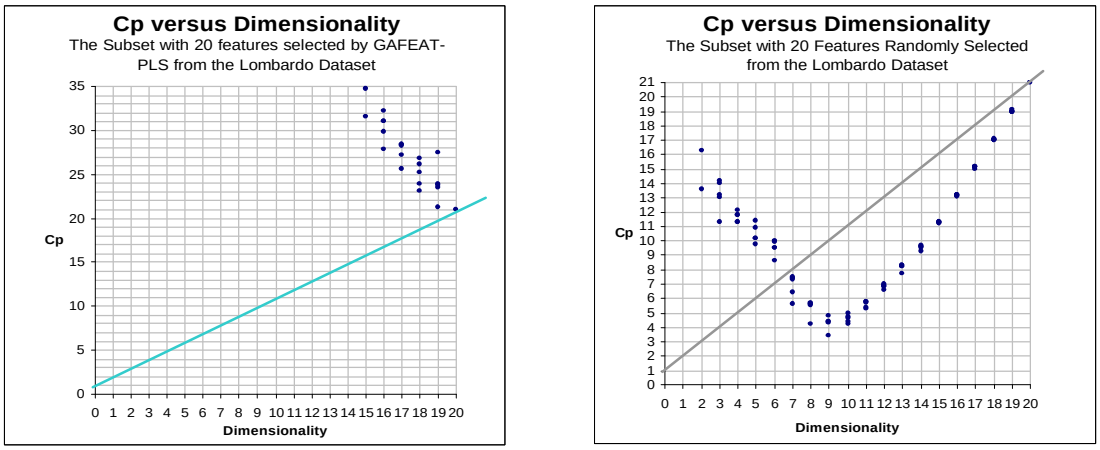
The other well-known feature selection criterion for linear regression is the Mallow's Cp [125]. The Mallow's Cp statistic also measures how well a model fits data, but with a penalty for adding extra independent variables. Since predicted values obtained from a subset regression model are biased, the mean square error of prediction consists of two components: the variance of prediction arising from estimation, and a bias arising from the deletion of variables. Mallow's Cp statistics is defined as

$$Cp = \frac{SSEp}{\hat{\sigma}^2} + (2p - n)$$

where $\hat{\sigma}^2$ is the estimate of the variance of the random error, which is calculated from the full regression model. The expected value of the **Cp** is **p** when there is no bias in the model. Therefore, it is wanted to select a model in which the value of **Cp** is close to the number of terms, including the constant term, in the model.

Using the Mallow's Cp statistic as a feature selection criterion, the best subsets regression of Minitab release 12.2 and all-subsets regression of S-PLUS 4.5 are also used to evaluate feature subsets selected by GAFEAT-PLS from the Lombardo and HIV_{rt} datasets up to 30 features. It has been found that no subsets of those subsets (with up to 20 features) selected by GAFEAT-PLS performed better than the original feature subsets based on the Mallow's Cp statistic. As an example to this computational experiment, the Cp plots of a subset with 20 features selected by GAFEAT-PLS and a subset with 20 feature randomly selected from the Lombardo dataset are presented in Figures 9.19-A and 9.19-B, respectively. The values of Cp of the best 5 models of each dimensionality versus dimensionality are plotted. It can be clearly seen from Figure 9.19-A that the best dimensionality for the subset of GAFEAT-PLS is 20 based on Cp criterion, which means

that none of feature subsets are performed better than the original subset selected by GAFEAT-PLS. The best dimensionality for the randomly subset selected subset is 7.

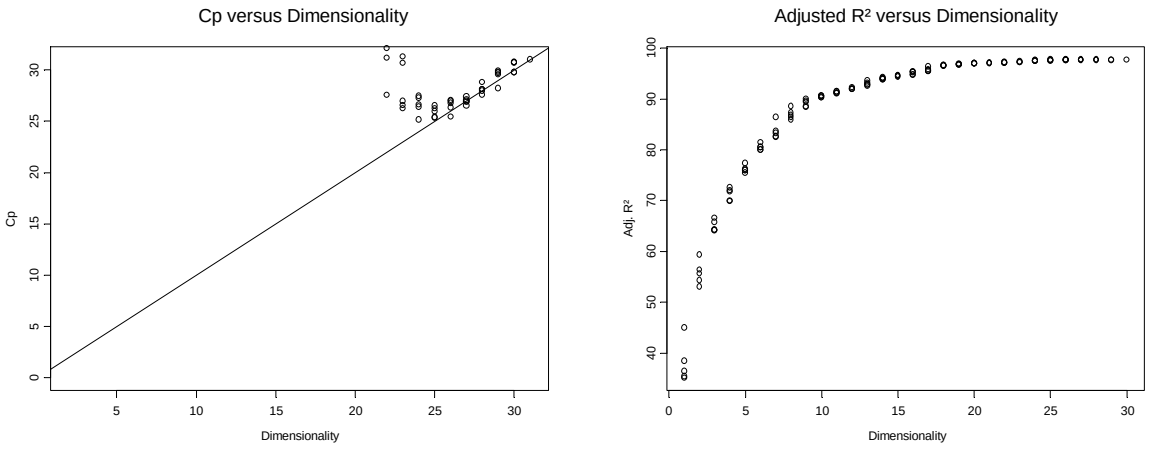


(A) **(B)**

Figure 9.19 The Best Subset Regression Results Cp versus Dimensionality **(A)** the Subset with 20 Features Selected by GAFEAT-PLS from the Lombardo Dataset **(B)** the Subset with 20 Features Randomly Selected from the Lombardo Dataset

GAFEAT-PLS was executed to select 10, 20, 30, 40, and 50 features from the HIV_{rt} and Lombardo datasets in order to find the most informative subsets with the minimum number of features. Therefore, the obtained best subsets with minum number of features are relative to the number of GAFEAT-PLS runs carried out for the datasets. Hovewer, it is always possible to zoom in to determine the exact subset with minimum number of features. Our experiments show that even though the selected feature subsets consist of more features than the fine-tuned (the actual optimal) feature subsets, the predictive performance of the models were not effected severely. GAFEAT-PLS feature selection on the expanded Lombardo dataset in section 9.5.6 is a example since the size

of the optimal feature subset is 20 but subsets with sizes of 20, 30, 40, and 50 features also give close prediction error (Q^2) based on a 100 bootstrap validation. All-subsets regression of S-PLUS 4.5 are used to evaluate the subset with 30 features selected by GAFEAT-PLS from the expanded Lombardo dataset based on both adj.R^2 and Mallow's Cp criteria. These results are presented in the Figure 9.20. It can be clearly seen from Figures 9.20-A and 9.20-B that even though a lower dimensional regression models exists based on both adj.R^2 and the Mallow's Cp criteria, their performances are very close to each other.



(A) **(B)**

Figure 9.20 The Best Subset Regression Results of the Subset with 30 Features Selected by GAFEAT-PLS from the Expanded Lombardo Dataset **(A)** Cp versus Dimensionality **(B)** Adj.R^2 versus Dimensionality

CHAPTER 10

GAFEAT-LL: Genetic Algorithms with Local Learning for Feature Selection

The traditional approach to supervised learning is global modeling describing the relationship between the input and output with an analytical function over the whole input domain [208]. For example, neural networks, Support Vector Machines, and PLS are global modeling techniques. One of the advantages of the global models is that after deriving a global model, there is no need to save the training data for prediction; therefore, it requires a small memory for the prediction on new data. The other advantage is that global models are very fast in the prediction phase. On the other hand, the global modeling approaches are typically slow and analytically intractable when they are applied to the highly nonlinear and complex input/output relations [208]. For this reason, local modeling has become an alternative to global modeling in the research community. Local modeling is a *divide-and-conquer* strategy, which is based on attacking a complex problem by dividing the problem into simpler problems whose solutions can be combined to obtain a solution to the original problem [208].

In this chapter, a local learning algorithm is integrated as an evaluation function into the genetic algorithm with the unique list representation developed in section 8.3.3. First of all, some information about the local learning algorithm is presented. Second, a literature review related to feature selection with evolutionary algorithms with LL is presented. In later sections, the details of the proposed Genetic Algorithm with Local

Learning (GAFEAT-LL) and its performance on two QSAR datasets (the Lombardo and HIV_{rt}) will be presented.

10.1 Local Learning

Memory-based learning refers to a family of algorithms that process training data until a query needs to be answered [142]. These algorithms store all of the training data in a memory and wait for a query to do further calculation to answer this particular query. The query is answered by finding *relevant* training data points. Relevance is generally calculated using a distance function. This type of learning is also called Lazy Learning [142]. Some of the memory-based learning algorithms find a set of nearest neighbors around a query point and answer the query by fitting a parametric function in its neighborhood. This type of memory-based learning is referred to as Local Learning (LL) [142, 144, 209]. Nearest neighbor, weighted average, and locally weighted regression are examples to local learning methods [142]. These methods, unlike to global modeling methods such as PLS, are '*locally parametric*' and do not produce a 'visible' model of the data. Instead they make predictions based on local models generated on a query point basis. Although LL is a non-parametric regression technique, it does have several 'parameters' that must be tuned in order to obtain good predictive results.

One of the most important is the notion of neighborhood. Given a query point **q**, it needs to be decided which training cases will be used to fit a local polynomial around the query point. This procedure involves defining a distance metric over the multi-dimensional space defined by the input variables. With this metric, a distance function can be specified, which allows finding the nearest training cases of any query point.

There are other issues to be addressed, such as weighting of the variables within the distance calculations (feature weighting) and the number of training cases (**L**) that enters the local fit (known as the bandwidth selection problem, generally chosen as 3 or 5) [144]. After determining the bandwidth specification, it is required to weight the contribution of the training cases within the bandwidth. This is usually accomplished by a weighting function or kernel function. Weighing the data can be viewed as re-emphasizing the relevant instances and de-emphasizing irrelevant instances. In other words, nearer instances to the query point contribute more into local fit [142, 144].

In this dissertation, the distance function is the Euclidean distance. A distance **$d(x^i,q)$** between the query point **q** and a data point **i** (between their input features) is defined by

$$d(x^i,q) = \sqrt{\sum_{j=1}^{N}(x_j^i - q_j)^2} = \sqrt{(x^i - q)^T(x^i - q)}$$

where, N is the number of features and x_j^i is the **j**th component of the vector $\mathbf{x^i}$.

The weighting function (kernel function) K() is defined by

$$K(d) = d^{-\theta}$$

where d is the distance $d(x^i,q)$ and θ is a user defined weight factor that allow the user to influence the contribution of the nearer data points ($\theta > 0$). A value of 1.3 for the parameter θ is used for all calculations in this dissertation. The outcome ($\hat{y}_q$) for a query point **q** is estimated by local learning model from the target outcomes of its **L** nearest neighbor (y_i) according to:

$$\hat{y}_q = \frac{\sum_{i=1}^{L} y_i K(d(x^i, q))}{\sum_{i=1}^{L} K(d(x^i, q))}$$

A value 5 for the parameter **L** (the number of training cases that enters the local fit) is used for all calculations in this dissertation.

10.2 Feature Selection with Evolutionary Algorithms and Local Learning

Embrechts et al. [144, 209] propose a novel approach for the supervised training of regression systems. The proposed method, **S**upervised **S**caled **R**egression **C**lustering with **G**enetic **A**lgorithms (SSRCGA), relies on a GA supervised clustering algorithm with local learning. The chromosomes of the GA represent the coordinates of the clusters. For instance, if the dimensionality of the data is **D** and number of predetermined cluster centers is **K**, the number of genes in a chromosomes will be **D***__K__. The GA is a floating-point GA that uses arithmetic crossover and uniform mutation [8]. SSRCGA tries to minimize a fitness function **F**, which is defined by

$$F = (J \pm \gamma N_E) + \alpha M_R ,$$

where J is the classical cluster dispersion measure, γ is a "dummy cluster" penalty/bonus factor, N_E is the number of empty clusters, M_R is the penalty factor proportional to the total regression error, and α is the regularization parameter. The SSRCGA starts out with a relatively large predetermined number of clusters and allows the number of clusters to vary by adding a penalty/bonus term for empty clusters. A cluster is empty when it has no members. Empty clusters do not effectively contribute to cluster dispersion and it

depends on the particular application whether a penalty or bonus is the more appropriate approach. For each cluster, local learning method is applied to calculate the outcome. The clustering itself is influenced by the results of the local learning model (M_R). The regularization parameter α can be problem dependent and needs to be specified by the user or determined by trial and error. In SSRCGA, the local learning is supervised in the sense that the prediction quality is incorporated as a penalty term added to the cost function of the genetic algorithm. Adaptive dimension scaling is also added into the SSRCGA. This is implemented by adding a number of genes, representing scaling factors, into the chromosomes corresponding to the dimensionality (**D**). Each dimension (feature) is multiplied by its corresponding scaling factor in order to discourage irrelevant features. The sum of the scaling factors is normalized to unity to prevent a trivial solution. The GA adaptively adjusts appropriate scaling factors and the most relevant features for the dataset under scrutiny are the ones with the larger scaling factor. The SSRCGA has advantages compared to traditional neural network approaches. These advantages are: i) the simplicity of the idea; ii) the flexibility of its implementation by allowing the user to modify the cost function and the penalty terms (e.g. the misclassification error measure); iii) a straightforward methodology for feature selection via scaling; and iv) a good general performance, even for high-dimensional datasets. On the other hand, the SSRCGA has some disadvantages compared to traditional neural network approaches. These disadvantages are: i) possible excessive demands on computing time and memory; ii) poor scaling of the speed of the algorithm with the number of data points; and iii) the ad-hoc problem choice for problem dependent regularization parameters.

Zheng and Tropsha [100] developed an automated variable selection method for quantitative structure-activity relationship (QSAR) based on an integration of k-Nearest Neighbor (kNN) principle into simulated annealing as a cost function. The basic idea behind this method (kNN QSAR) is that similar compounds display similar profiles of pharmacological activities. The activity of each compound is predicted as an average activity of **k** most chemically similar compounds from the dataset. The kNN QSAR algorithm searches for both optimum **k** value and an optimal subset with a predefined number of descriptors, which together build a QSAR model with the best predictive ability in terms of leave-one-out prediction error. Since the number of features is predefined, it needs to be optimized by setting to different values in several different runs. They demonstrated the robustness of the QSAR models by comparing them to those derived from randomized datasets.

K-nearest-neighbors (kNN) methods are commonly used for analyzing datasets that cannot be assumed to have a normal distribution [210]. Pure kNN classifiers cannot perform well on datasets, which have noisy input data, outliers, and correlated features. Weighting the feature axes according to their relative importance can help kNN classifiers. If there is a priori knowledge of the relative contribution of the each feature, then features can be weighted accordingly. If there is no such prior information available, it is impossible to weight features. Genetic algorithms have been widely used to learn feature weights for the kNN classifiers [210-212]. In these implementations, the length of a chromosome is equal to the number of features in the dataset. A chromosome is a vector consisting of a real-valued weight for each of the features. The GA is used to

assign weights to features in order to discourage the irrelevant and weakly relevant features and reward the relevant features.

Raymer et al. [210, 211] have applied genetic algorithms to the problem of feature selection. In their work, the genetic algorithm performs feature selection and extraction in combination with a k-nearest-neighbors (kNN) classifier, which is used to evaluate the classification performance of each subset features selected by the GA. The GA is used to perform simultaneous feature selection and feature extraction. The GA uses both a feature weight vector and a masking (selection) vector on its chromosomes. A feature selection vector consists of a single bit for each feature, with a '1' indicating that the feature is included in the kNN classification, and a '0' indicating that it is omitted. A feature weight vector consists of a real-valued weight for each of the features. Each feature is multiplied by both its weight and mask values prior to classification by the kNN classifier. One of the advantages of this method is that the GA can test the effect of eliminating a feature completely from the classification by setting its mask value to zero without reducing the associated feature weight to zero. This prevents losing the previously learned weights and makes the search efficient and faster.

10.3 The Proposed GAFEAT-LL

In the proposed Genetic Algorithm with Local Learning (GAFEAT-LL), local learning algorithm is integrated as a cost function into the GA with unique list representation developed in section 8.3.3. Given a dataset containing **T** features, each chromosome represents a legal subset containing **N** features. In this representation, a chromosome is as an integer array with size **N**, where **N** is the predetermined number of

features to be selected out of total **T** features. Each gene represents the number for corresponding feature in the dataset. The details of the genetic algorithm are explained in section 9.3. The only difference now is the cost function where PLS regression is replaced with the Local Learning algorithm explained in section 10.1.

10.4 Computational Evaluation of GAFEAT-LL

Two QSAR datasets (the Lombardo and HIV_{rt} datasets), which were used to evaluate the performance of GAFEAT-PLS, have been used to evaluate the performance of GAFEAT-LL. The first step is to use objective feature selection (e.g. removing non-changing, highly inter-correlated (*cousin features*), and 4σ outlier features) to remove features that contain redundant, minimal or distorted information. StripMiner™ [4] is employed to perform the objective feature selection.

Since GAFEAT-LL conducts a search for a good feature subset using the local learning algorithm itself as part of the fitness function, the accuracy estimation of the LL models can be overly optimistic (over-fitting). Therefore, another predictive methods, namely the SVM regression is used for constructing predictive models for the selected subsets by GAFEAT-LL and accuracy estimations of SVM regression models (validation errors) are compared with that of LL models. PLS regression method is also used for modeling the selected feature subset to see how selected subsets effects the linear method.

10.4.1 The Lombardo Dataset

After pre-processing, 309 descriptive features remained in the Lombardo dataset. The StripMiner™ [4] program is employed to construct predictive models (the LL, PLS, and SVM models) of the Lombardo dataset with 309 original features, and these models are validated based on 100-bootstrap samples by leaving out 6 molecules in the validation sets. The LL models described in this dissertation used five nearest neighbors and weight factor θ is set to 1.3 for all calculations. The PLS regression models used 4 latent variables for all calculations. These validation results are presented in the Table 10.1.

Table 10.1 100-bootstrap Validation of Full Lombardo Dataset with PLS, SVM, and LL

Learning Models	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
LL	0.5563	0.5531	0.3665	0.3689
PLS	0.9228	0.9228	0.2853	0.2858
SVM	0.9846	0.9822	0.3098	0.3105

GAFEAT-LL is applied on the Lombardo dataset to select a good feature subset with a minimum number of features. The parameters of GAFEAT-LL are presented in the Table 10.2. Since the number of feature to be selected is predetermined, a several of GAFEAT-LL runs are performed to find the most informative feature subset with the minimum number of descriptive features. GAFEAT-LL is run to select 10, 20, 30, 40, and 50 features from the Lombardo dataset with 309 features. Each selected feature subset is modeled by the LL, PLS, and SVM algorithms and these models are further validated with 100-bootsrap samples by leaving 56 molecules in training sets and 6 molecules in

the validation sets. The feature subset validation results for the LL, PLS, and SVM models are presented in the Tables 10.3, 10.4, and 10.5, respectively.

Table 10.2 Parameters of GAFEAT-LL for the Lombardo Dataset

Population Size	Crossover Probability	Mutation Probability	Maximum # of Generation	Number of Latent Variables
100	0.90	0.02	1000	4
# of Bootstraps in the Fitness Evaluation	# of Molecules in the Training Set	#of Molecules in the Validation Set	Learning Weight Factor (θ)	
20	56	6	1.3	

Table 10.3 LL 100-bootstrap Validation Results for Feature Subsets Selected by GAFEAT-LL from the Lombardo Dataset

Number of Selected Features	**Training Error**		**Validation Error**	
	r^2	**R^2**	**q^2**	**Q^2**
10	0.8957	0.8878	0.0981	0.1037
20	0.8645	0.8605	0.1178	0.1204
30	0.8867	0.8782	0.0962	0.1031
40	0.8968	0.8770	0.0998	0.1140
50	0.8884	0.8775	0.1212	0.1301

Table 10.4 PLS 100-bootstrap Validation Results for Feature Subsets Selected by GAFEAT-LL from the Lombardo Dataset

Number of Selected Features	**Training Error**		**Validation Error**	
	r^2	**R^2**	**q^2**	**Q^2**
10	0.8284	0.8284	0.2267	0.2275
20	0.8619	0.8619	0.2140	0.2157
30	0.8933	0.8933	0.2185	0.2223
40	0.9272	0.9272	0.1964	0.1984
50	0.9171	0.9171	0.2191	0.2235

Table 10.5 SVM 100-bootstrap Validation Results for Feature Subsets Selected by GAFEAT-LL from the Lombardo Dataset

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.8371	0.8363	0.3008	0.3016
20	0.8494	0.8430	0.2412	0.2426
30	0.8851	0.8781	0.2392	0.2392
40	0.9197	0.9116	0.2237	0.2255
50	0.904	0.8906	0.2278	0.2291

Figure 10.1 shows the number of features selected by GAFEAT-LL versus 100-bootstrap validation errors of the LL, PLS, and SVM models of the corresponding selected feature subsets in terms of Q^2 statistics. Although the performances of the SVM and PLS regression models constructed with the feature subsets selected by GAFEAT-LL are similar and better than the dataset with all features, the performances of the LL models are significantly better than the PLS and SVM regression models; indicating possibly a bias (i.e., overfitting) in the selected features.

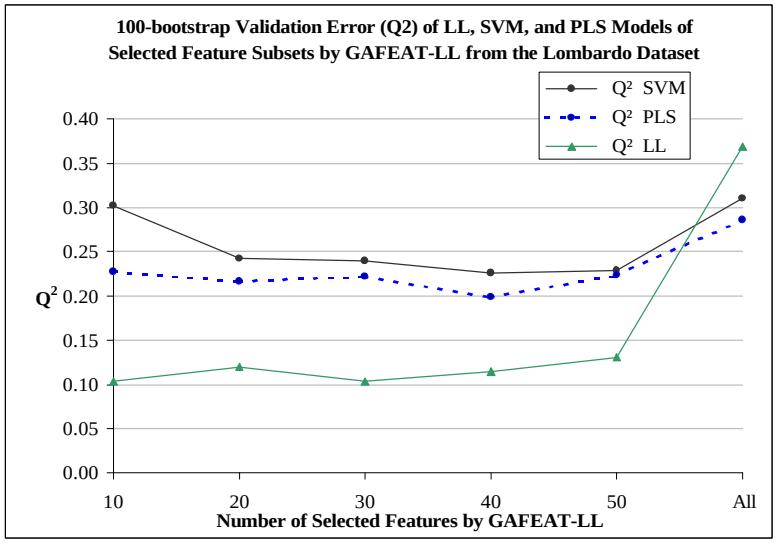


Figure 10.1 100-bootstrap Validation Errors (Q^2) of the LL, SVM, and PLS Models of Selected Feature Subsets by GAFEAT-LL from the Lombardo Dataset

It has been hypothesized that a good feature subset selected by GAFEAT-LL should give similar results for the SVM regression. Since LL can be thought of as a non-linear method, GAFEAT-LL feature selection method selects features describing the non-linear relationship existing in the dataset. The results show that the PLS models have a slightly better performance than the SVM models, although the PLS regression is a linear modeling approach. These results may indicate that GAFEAT-LL feature selection method simply overfit the data. Therefore, a randomization test will be applied to GAFEAT-LL feature selection in section 10.5 in order to verify whether the feature subsets selected by GAFEAT-LL model noise instead of the underlying information in the dataset.

10.4.2 The HIV_{rt} Dataset

After pre-processing, 230 descriptive features left in the HIV_{rt} dataset. The StripMiner™ [4] program is employed to construct predictive models (the LL, PLS and SVM models) of the HIV_{rt} dataset with 230 original features. The obtained LL, PLS and SVM models are further validated based on 100-bootstrapping samples by leaving 58 molecules in training set and 6 molecules in the validation sets. These validation results are presented in the Table 10.6.

Table 10.6 100-bootstrap Validation of HIV_{rt} Full Dataset with LL, PLS, and SVM

Learning Models	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
LL	0.6233	0.6228	0.4956	0.5045
PLS	0.8129	0.8129	0.3354	0.3368
SVM	0.9276	0.9159	0.3690	0.3718

GAFEAT-LL is applied to select a good feature subset with the minimum number of features on the HIV_{rt} dataset. The parameters of GAFEAT-LL used for the HIV_{rt} dataset is presented in the Table 10.7. Several different GAFEAT-LL runs were executed in order to find the most informative feature subset with the minimum number of features.

Table 10.7 Parameters of GAFEAT-LL for the HIV_{rt} Dataset

Population Size	Crossover Probability	Mutation Probability	Maximum # of Generation	Number of Latent Variables
100	0.90	0.02	1000	4
# of Bootstraps in the Fitness Evaluation	# of Molecules in the Training Set	#of Molecules in the Validation Set	Learning Weight Factor (θ)	
20	58	6	1.3	

GAFEAT-LL was executed to select 10, 20, 30, 40, and 50 features from the HIV_{rt} dataset with 230 features. Each selected feature subset is modeled by the LL, PLS, and SVM algorithms and these models are further validated with 100-boostrap samples by leaving 58 molecules in training set and 6 molecules in the validation set. The feature subset validation results for the LL, PLS, and SVM models are presented in the Tables 10.8, 10.9, and 10.10, respectively.

Table 10.8 LL 100-bootstrap Validation Results for Feature Subsets Selected by GAFEAT-LL from the HIV_{rt} Dataset

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.7877	0.7827	0.1827	0.1828
20	0.8278	0.8252	0.1663	0.1686
30	0.8141	0.8126	0.2042	0.2044
40	0.8123	0.8122	0.2097	0.2111
50	0.7845	0.7802	0.2162	0.2172

Table 10.9 PLS 100-bootstrap Validation Results for Feature Subsets Selected by GAFEAT-LL from the HIV_{rt} Dataset

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.5073	0.5073	0.6319	0.6526
20	0.6522	0.6522	0.5041	0.5275
30	0.7049	0.7048	0.4887	0.5030
40	0.7458	0.7458	0.4121	0.4248
50	0.7043	0.7043	0.4494	0.4682

Table 10.10 SVM 100-bootstrap Validation Results for Feature Subsets Selected by GAFEAT-LL from the HIV_{rt} Dataset

Number of Selected Features	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
10	0.9706	0.9692	0.2232	0.2237
20	0.9640	0.9611	0.1973	0.2019
30	0.9814	0.9789	0.2107	0.2119
40	0.9281	0.9219	0.2224	0.2224
50	0.9981	0.9981	0.1766	0.1841

Figure 10.2 shows the number of selected features by GAFEAT-PLS versus the 100-bootstrap validation errors of the LL, PLS and SVM models of the corresponding selected feature subsets in terms of Q^2 statistics.

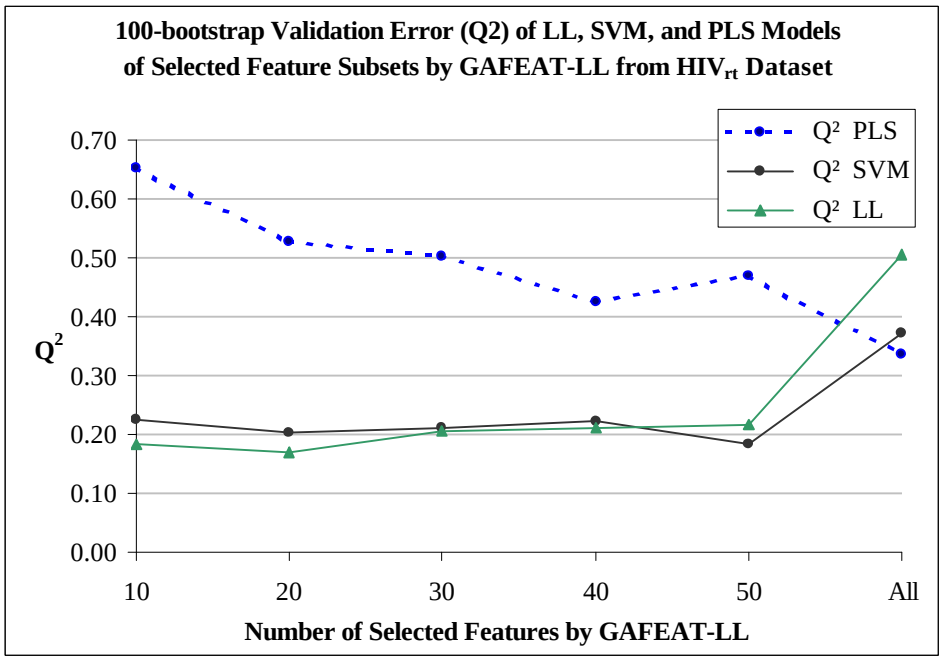


Figure 10.2 100-bootstrap Validation Errors (Q^2) of the LL, SVM, and PLS Models of Selected Feature Subsets by GAFEAT-LL from the Lombardo Dataset

It is clear from these results that PLS regression method is not able to model the dataset with the features selected by GAFEAT-LL, most likely because PLS is a linear regression method. On the other hand, the performances of the SVM and LL models with the selected feature subsets are very similar and better than the models with the all features. Before, analyzing the performance of GAFEAT-LL in detail, the validation of GAFEAT-LL is performed with a randomization test in the following section.

10.5 Computational Validation of GAFEAT-LL with a Randomization Test

As explained in section 9.4.4, like GAFEAT-PLS, the validation (robustness) of GAFEAT-LL method is examined with a standard hypothesis testing where the results of a real dataset is compared to those of the dataset with randomly shuffled response values. The randomization test validation of GAFEAT-LL is performed on the Lombardo and HIV_{rt} datasets.

10.5.1 The Lombardo Dataset

Ten randomization trials were performed in which the biological activity data was randomized with respect to the dataset. A full GAFEAT-LL feature selection method is performed for each randomized dataset. Randomization of data, GAFEAT-LL feature selection, and subsequent model evaluation are performed to assess the statistical validity of QSAR models and GAFEAT-LL feature selection method. Each selected feature subset is modeled using the LL algorithm (with 5 nearest neighbors and weighting factor θ 1.3) and the predictive quality of the resulting models is measured based on 100 bootstrap samples leaving 56 molecules in the training set and 6 molecules in the validation set. Figures 10.3 and 10.4 show the results of randomization test of GAFEAT-LL feature selection method on the Lombardo dataset.

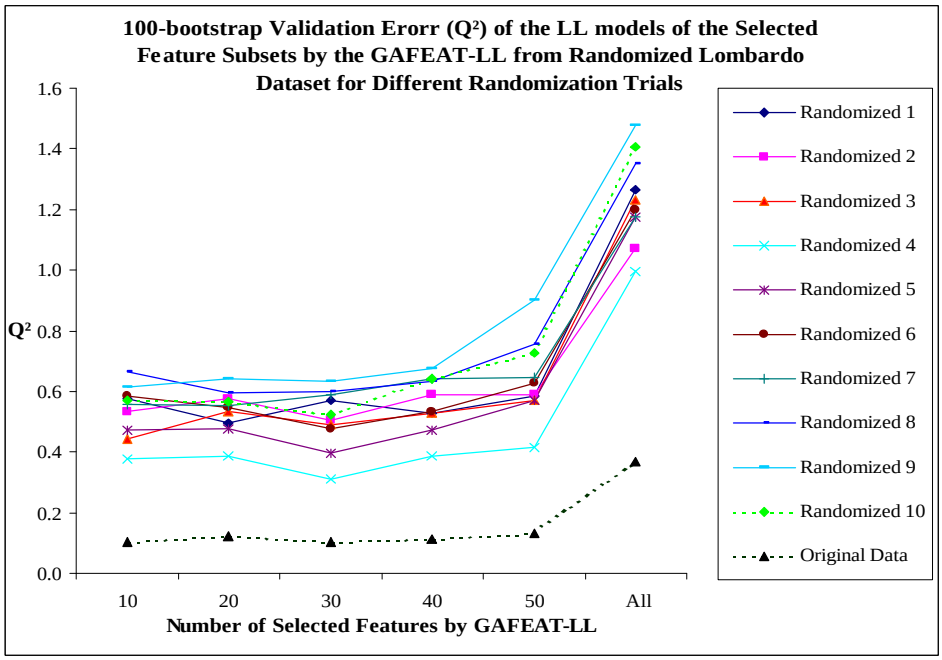


Figure 10.3 100-bootstrap Validation Errors (Q^2) of the LL Models of Feature Subsets Selected by GAFEAT-LL from Randomized Lombardo Dataset for Different Randomization Trials

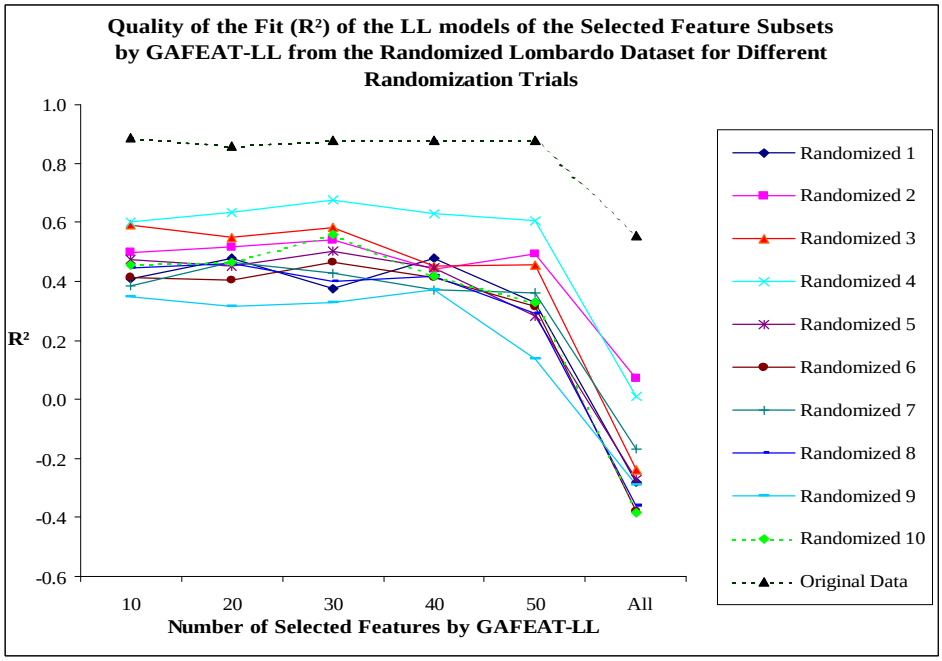


Figure 10.4 Quality of the Fit (R^2) of the LL Models of Selected Feature Subsets by GAFEAT-LL from the Randomized Lombardo Dataset for Different Randomization Trials

It appears from Figures 10.3 and 10.4 that in each randomization test the randomized data give much higher Q^2 and much lower R^2 values than those of the original data. As explained in section 9.4.4, the validation (robustness) of GAFEAT-LL method is carried out by standard hypothesis testing where the results for the real dataset are compared to those of randomized datasets. The hypothesis testing results based on R^2 and Q^2 are presented in Tables 10.11 and 10.12, respectively. Z score and P value represent the test statistics and its probability (i.e., the smallest probability leading to rejection of the null hypothesis) respectively.

Table 10.11 Standard One-Tail Hypothesis Testing of GAFEAT-LL on the Lombardo Dataset Based on R^2

Number of Selected Features	Original Data	Mean of the Randomization Trials	Std. Dev. of the Randomization Trials	Z score	P value
10	0.8878	0.4627	0.0826	5.1452	~0
20	0.8605	0.4731	0.0845	4.5863	0.0000
30	0.8782	0.4856	0.1067	3.6780	0.0001
40	0.877	0.4447	0.0733	5.8971	~0
50	0.8775	0.3602	0.1297	3.9897	0.0000
All	0.5531	-0.2297	0.1568	4.9926	0.0000

Table 10.12 Standard One-Tail Hypothesis Testing of GAFEAT-LL on the Lombardo Dataset Based on Q^2

Number of Selected Features	Original Data	Mean of the Randomization Trials	Std. Dev. of the Randomization Trials	Z score	P value
10	0.1037	0.5396	0.0860	-5.0674	~0
20	0.1204	0.5365	0.0706	-5.8936	~0
30	0.1031	0.5091	0.0981	-4.1384	0.0000
40	0.114	0.5636	0.0904	-4.9722	0.0000
50	0.1301	0.6397	0.1310	-3.8899	0.0001
All	0.3689	1.2343	0.1469	-5.8916	~0

Based on P values, it can be concluded with 90 percent confidence that GAFEAT-LL models on original datasets give consistently higher R^2 and lower Q^2 than those of GAFEAT-LL models obtained on the ten different randomized datasets.

10.5.2 The HIV_{rt} Dataset

Ten randomization trials were performed in which the biological activity data was randomized with respect to the dataset. A full GAEFAT-LL feature selection and modeling method is performed for each randomized HIV_{rt} dataset. Each selected feature subset is modeled using the LL algorithm (with 5 nearest neighbors and weighting factor θ 1.3) and the predictive quality of the resulting models is measured based on 100 bootstrap samples leaving 6 molecules in the validation set. Figures 10.5 and 10.6 show the results of randomization testing of the GAFEAT-LL feature selection method on the HIV_{rt} dataset.

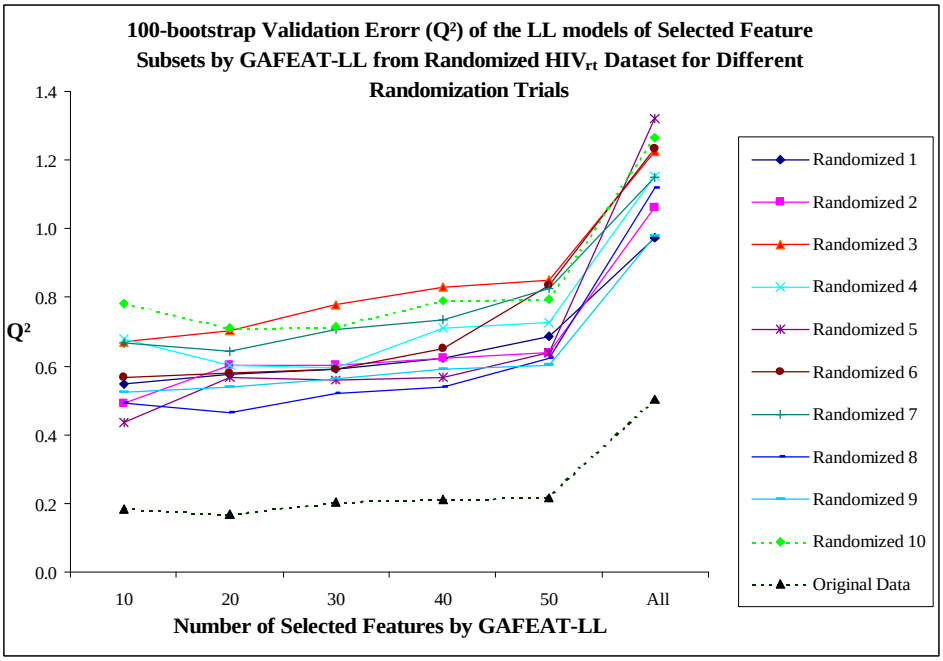


Figure 10.5 100-bootstrap Validation Errors (Q^2) of the LL Models of Selected Feature Subsets by GAFEAT-LL from the Randomized HIV_{rt} Dataset for Different Randomization Trials

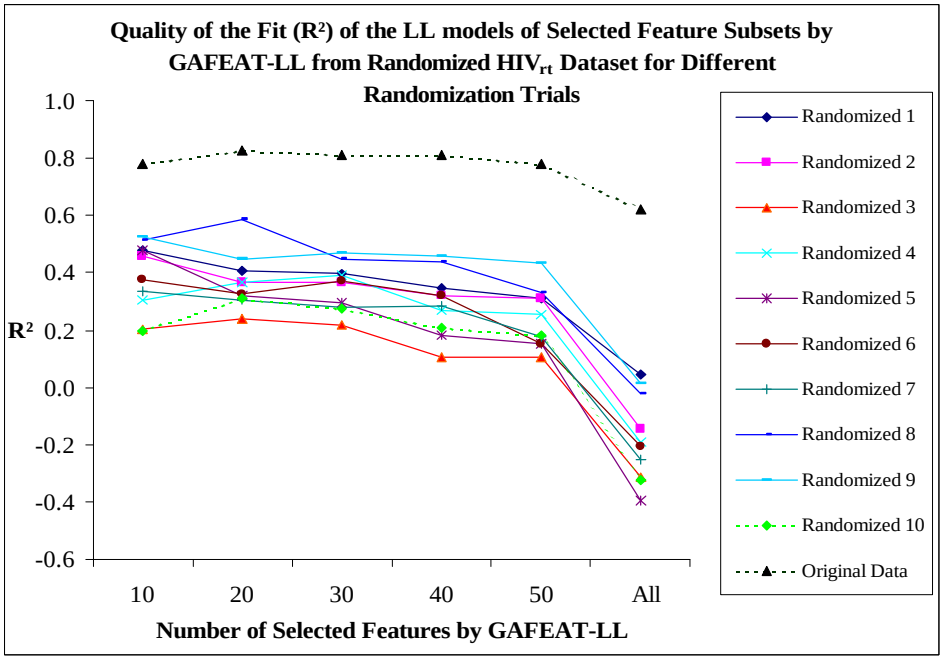


Figure 10.6 Quality of the Fit (R^2) of the LL Models of Selected Feature Subsets by GAFEAT-LL from the Randomized HIV_{rt} Dataset for Different Randomization Tests

It appears from the Figures 10.5 and 10.6 that in each randomization test the randomized data give much higher Q^2 and much lower R^2 values than those of the original data. The hypothesis testing results based on R^2 and Q^2 are presented in the Tables 10.13 and 10.14, respectively. Z score and P value represent the test statistics and its probability (i.e., the smallest probability leading to rejection of the null hypothesis).

Table 10.13 Standard One-Tail Hypothesis Testing of GAFEAT-LL on the HIV_{rt} Dataset Based on R^2

Number of Selected Features	Original Data	Mean of the Randomization Trials	Std. Dev. of the Randomization Trials	Z score	P value
10	0.7827	0.3863	0.1240	3.1962	0.0007
20	0.8252	0.3675	0.0965	4.7457	0.0000
30	0.8126	0.3513	0.0814	5.6676	~0
40	0.8122	0.2933	0.1104	4.7011	0.0000
50	0.7802	0.2391	0.1037	5.2153	~0
All	0.6228	-0.1796	0.1509	5.3174	~0

Table 10.14 Standard One-Tail Hypothesis Testing of GAFEAT-LL on the HIV_{rt} Dataset Based on Q^2

Number of Selected Features	Original Data	Mean of the Randomization Trials	Std. Dev. of the Randomization Trials	Z score	P value
10	0.1828	0.5846	0.1095	-3.6697	0.0001
20	0.1686	0.5977	0.0740	-5.8005	~0
30	0.2044	0.6214	0.0829	-5.0303	~0
40	0.2111	0.6657	0.0967	-4.7023	0.0000
50	0.2172	0.7215	0.0968	-5.2069	~0
All	0.5045	1.1475	0.1177	-5.4650	~0

Based on P values, it can be concluded with a 90 percent of confidence that GAFEAT-LL models on the original dataset give consistently higher R^2 and lower Q^2 than those of GAFEAT-LL models obtained from ten different randomized datasets.

10.6 Final Remarks on GAFEAT-LL Feature Selection

The results of the randomization tests of GAFEAT-LL on the Lombardo and HIV_{rt} datasets showed the robustness of the LL models constructed with the feature subsets selected by GAFEAT-LL as well as the robustness of the GAFEAT-LL feature selection method. Computational results of GAFEAT-LL on the Lombardo and HIV_{rt} datasets showed that GAFEAT-LL was able to selected good feature subsets that resulted more predictive models than that of that of the model with all features.

The results of the Lombardo dataset in section 10.4.1 showed that the PLS models of the feature subsets selected by GAFEAT-LL performed slightly better than those of the SVM models, and that the performances of the LL models were significantly better than the PLS and SVM regression models. According to the performance of GAFEAT-LL on HIV_{rt} dataset, the SVM and LL models performed similar but the performances of the PLS models were significantly worse than those of the SVM and LL models. This result would be expected since LL method can be thought of as a non-linear method and the GAFEAT-LL selects feature subsets describing the non-linear characteristics of the dataset. It is also known that the molecules in the HIV_{rt} dataset are clustered in to five classes; therefore, this dataset is known to more non-linear. The basic idea behind the local learning method is that structurally similar compounds should have similar biological activities [100]. Based on the performances of GAFEAT-LL on the Lombardo and HIVrt datasets, it can be concluded that GAFEAT-LL feature selection method works better in datasets in which several classes of chemical compounds are encountered.

Since the PLS and LL algorithms are used as a fitness function by GAFEAT-PLS and GAFEAT-LL, respectively, the prediction errors of the resulting PLS and LL models

can be overly optimistic (e.g. bias to the specific method). For this reason, all three methods are compared based on the performance of the SVM models on the feature subsets selected by the corresponding feature selection method. It is also worth mentioning that GAFEAT-PLS selects features based on a linear criterion, since PLS regression is a linear method. On the other hand, GAFEAT-LL selects features based on a non-linear criterion. If the relationship between the selected features and response variable is linear, then it is expected that SVM regression, which is a nonlinear model, will converge to the linear model.

Comparison of the feature selection methods on the Lombardo and HIV_{rt} datasets based on the best SVM models are presented in the Tables 10.15 and 10.16, respectively. The results were calculated based on 100 bootstrap samples. GAEFAT-PLS with INLR method performs better than the GAFEAT-PLS and GAFEAT-LL methods. Although the performance of GAFEAT-LL on the Lombardo dataset is significantly worse than GAFEAT-PLS methods, its performance on the HIV_{rt} dataset is comparable to GAFEAT-PLS. In conclusion, based on results of the Lombardo and HIV_{rt} datasets, three feature selection methods (GAFEAT-PLS, GAFEAT-PLS with INLR method, and GAFEAT-LL) worked well for selecting feature subsets that results more predictive models than that of the model with all features.

Table 10.15 Comparison of the Feature Selection Methods on the Lombardo Dataset Based on the Best SVM Models

Feature Selection Method	Number of Selected Features	Training Error		Validation Error	
		r^2	R^2	q^2	Q^2
GA-PLS	30	0.9810	0.9795	0.0719	0.0733
GA-PLS with INLR	50	0.9934	0.9928	0.0504	0.0521
GA-LL	40	0.9197	0.9116	0.2237	0.2255

Table 10.16 Comparison of the Feature Selection Methods on the HIV_{rt} Dataset Based on the Best SVM Models

Feature Selection Method	Number of Selected Features	Training Error		Validation Error	
		r^2	R^2	q^2	Q^2
GA-PLS	30	0.9753	0.9749	0.1456	0.1470
GA-PLS with INLR	30	0.9554	0.9521	0.1336	0.1371
GA-LL	50	0.9981	0.9981	0.1766	0.1841

CHAPTER 11

Conclusions and Scope of Future Work

The contributions of this dissertation can be categorized into four main areas:

- The design of novel evolutionary algorithms for solving the Traveling Salesman Problem.
- The design of novel evolutionary algorithms for feature selection problem: i) genetic algorithm with floating-point representation, ii) genetic algorithm with unique list representation iii) evolutionary programming algorithm with unique list representation.
- A novel correlation based feature selection method (GAFEAT) and its hybridization with the sensitivity analysis.
- Applications of the developed evolutionary algorithms to predictive data mining problems, especially in drug design (QSAR) problems.

Novel genetic algorithms for the TSP with a modified Partially Mapped Crossover (PMX) operator were developed where the concept of PMX is carried further. The PMX was originally proposed by Goldberg and Lingle [1]. An algorithm for the TSP based on evolutionary programming was also developed. The proposed evolutionary programming algorithm, Evolutionary Programming with Constant Population (EPC), is different from the standard evolutionary programming algorithm in the sense that EPC always keeps a constant number of individuals in the population similar to genetic

algorithms and that EPC uses a selection scheme (i.e., simulated annealing) with a mutation operation.

Novel genetic algorithms for feature selection problem based on floating-point and unique list representations were developed. An evolutionary programming algorithm for feature selection was also developed based on the unique list representation. The computational results have demonstrated that the proposed evolutionary algorithms are capable of converging to optimal or near optimal solutions depending on the specific objective function criterion used. The proposed evolutionary algorithms for feature selection require a predetermined number of descriptive features as an input. The idea behind the fixing the number of features to be selected for each run of an evolutionary algorithm is to make search efficient. The proposed evolutionary algorithms can also be allowed to determine this parameter automatically (optimal dimension), but this strategy can potentially cause the algorithm to convergence prematurely. Since each dimension (subset size) has its global optima (the best subset), and the search spaces of the subsets are not equally represented in the total search space, the evolutionary algorithms have the tendency to converge to subsets with a few or with more features.

GAFEAT, a novel GA-based feature selection methodology, is based on the correlation matrix. The GA determines which descriptive features have the best correlation with the response, but have a relatively weak inter-correlation. The advantages of GAFEAT are (i) the generally robust subset of selected features, and (ii) that it scales linearly with the number of descriptive features (with only a weak dependency on the number of molecules in the dataset). The disadvantages of GAFEAT are the ad-hoc heuristics for determining the control parameters in the algorithm, and the

user generally does not know the right number of selected features. For QSAR studies selecting about 40 features, a conservative number, ensures that important features are included in the model.

A hybrid feature selection method, which combines GAFEAT selection method with neural network sensitivity analysis, was also proposed. The drawback of the neural network sensitivity analysis methodology is that the method is time consuming and does not scale-up well to very large data sets. The consideration of computational efficiency for large datasets favors a combination of methods. Applying GAFEAT for coarse feature selection followed by a sensitivity-based fine-tuning for feature selection demonstrated indeed that GAFEAT was picking features with valuable information content.

GAFEAT is independent on the learning algorithm and is used as a filter to conduct a search for a good feature subset using a correlation-based evaluation function. GAFEAT can be thought of as a filter method, which selects features based on the training data alone and does not take the biases of modeling algorithms into consideration. The main disadvantage of filter methods is that they totally ignore the effect of the selected feature subset on the performance of the learning algorithm. In order to take into account the biases of the modeling algorithms, Partial Least Square (PLS) regression has been integrated into the GA as a fitness function.

A typical QSAR predictive data-mining problem dataset is characterized by a large number of highly inter-correlated descriptive features (300-1000) for a relatively small number of molecules. PLS regression is a useful tool to model datasets in which the number of features exceeds the number of observations and/or a high level of multi-collinearity among those features exists. GAFEAT-PLS feature selection method was

applied onto two QSAR datasets (the Lombardo and HIV_{rt} datasets) in order to analyze its performance. GAFEAT-PLS was able to identify feature subsets for models with good prediction. GAFEAT and GAFEAT-PLS choose feature subsets based on linear criteria. If a dataset has underlying linear characteristics, the selected subsets by GAFEAT-PLS produce better predictive linear (e.g. PLS) as well as non-linear models (e.g. SVM). However, generally real datasets (e.g. QSAR datasets) can exhibit significant nonlinear characteristics, which cannot be properly accounted for with linear methods. Most of the real datasets exhibit nonlinear characteristics to some degree. A wide degree of non-linearity (from mild to severe) may exist between sets of variables, and the mildest non-linearities can be modeled by quadratic polynomials [195]. Berglund and Wold [192] proposed a simple way to develop nonlinear PLS models (**I**mplicit **N**onlinear **L**atent variable **R**egression) within the linear PLS framework. They point out that if a dataset has mild non-linear characteristics, the INLR method works well. Therefore, the INLR method was combined with the GAs for feature selection. The results of GAFEAT-PLS with the INLR showed that by introducing the quadratic non-linearities into the linear PLS framework, the selected feature subset resulted more predictive models.

If the non-linearities cannot be accounted for with a quadratic polynomial, the INLR approach fails. The severe non-linearities make data look discontinuous and clustered, and the data cannot be modeled by any single continuous model [195]. In order to take into the consideration more complex non-linearities in the feature selection phase, GA algorithms combined with a Local Learning algorithm (GAFEAT-LL). Local modeling is a *divide-and-conquer* strategy, which is based on attacking a complex problem by dividing the problem into simpler problems whose solutions is combined to

obtain a solution to the original problem [208]. Computational results have shown that the GAFEAT-LL feature selection method performed better for clustered datasets with many separate classes. It is worth noting that the selected feature subsets by GAFEAT-LL produced better predictive model with the non-linear modeling methods (e.g. SVM) since GAFEAT-LL selects the features based on non-linear criterion and linear model can only model the linear relationship. In this dissertation, the aim was to find good feature subsets with lower dimensionality, which gave similar results for the linear (e.g. PLS) and non-linear (e.g. SVM) predictive models. If the relationship between the predictor data (X) and response data (Y) is linear, then it is expected that non-linear model (SVM) can converge to a linear model. Therefore, the feature subsets selected by GAFEAT-PLS and GAFEAT-PLS with INLR method produced a better model with linear and non-linear methods.

The GA feature selection could be implemented with other methods, which could take into consideration the more severe non-linearities in data. One of these methods is the GIFI approach [188, 195, 213]. In the GIFI approach, each variable of the X data is divided into a number of bins, and each bin represents a new variable. In other words, each original variable is represented by a set of new variables. This new representation allows for a non-linear representation for the corresponding original variable. This approach can be used for the GAFEAT-PLS to select appropriate feature subsets to model non-linear datasets. A second approach is the combination of the GAs feature selection with non-linear methods such as kernel PLS and SVM regression. In this approach, it is essential to develop an efficient algorithm, which can be trained very fast,

since GAs are population-based algorithms and can be slow due to the large number of required objective function evaluation.

CITED LITERATURE

[1] D. E. Goldberg and J. Lingle, R., "Alleles, Loci, and the Traveling Salesman Problem," presented at an International Conference on Genetic Algorithms and Their Applications, J. J. Grefenstette Eds., Lawrence Erlbaum, Hillsdale, New Yersey, pp. 154-159, Pittsburg, PA, 1985.

[2] C. Hansch, A. Leo, D. Hoekman, and P. Li, "A Comprehensive Approach to Structure-Activity Relationships," presented at Solute/Solvent Interactions, pp. 105-137, Aberdeen Proving Ground, Maryland, 1992.

[3] M. J. Embrechts, F. Arciniegas, M. Ozdemir, C. M. Breneman, K. P. Bennett, and L. Lockwood, "Bagging Neural Network Sensitivity Analysis for Feature Reduction in QSAR Problems," presented at IJCNN-IEEE International Joint Conference in Neural Networks (July 14-19), IEEE Press, vol. 4, pp. 2478-2482, Washington, D.C., 2001.

[4] M. J. Embrechts, F. Arciniegas, M. Ozdemir, and M. Momma, "Scientific Data Mining with StripMiner™.," presented at 2001 SMCia Mountain Workshop on Soft Computing in Industrial Applications (June 25-27), M. J. Embrechts, H. F. VanLandingham, and S. Ovaska Eds., IEEE Press, pp. 13-16, Blacksburg - Virginia, 2001.

[5] M. Ozdemir, M. J. Embrechts, F. Arciniegas, C. M. Breneman, L. Lockwood, and K. P. Bennett, "Feature Selection for In-silico Drug design Using Genetic Algorithms and Neural Networks," presented at IEEE SMCia-01 Mountain Workshop on Soft Computing in Industrial Applications, M. J. Embrechts, H. F. VanLandingham, and S. Ovaska Eds., IEEE Press, pp. 53-57, Blacksburg - Virginia, 2001.

[6] R. Kohavi and G. H. John, "The Wrapper Approach," presented at Feature Selection for Knowledge Discovery and Data Mining, H. Liu and H. Motoda Eds., Kluwer Academic Publishers, pp. 33-50, 1998.

[7] R. Leardi, R. Boggia, and M. Terrile, "Genetic Algorithms as a Strategy for Feature Selection," *Journal of Chemometrics*, vol. 6, pp. 267-281, 1992.

[8] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3 ed. New York: Springer-Verlag, 1996.

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley Publishing Company, Inc., 1989.

[10] H. P. Schwefel, *Evolution and Optimum Seeking*. New York: John Willey and Sons, 1995.

[11] C. M. Fonseca and P. J. Fleming, "Genetic Algorithms for Multi-objective Optimization: Formulation, Discussion and Generalization," presented at the Fifth International Conference on Genetic algorithms, S. Forrest Eds., Morgan Kaufmann, pp. 141-153, San Mateo, California, 1993.

[12] K. Deb, "Genetic Algorithms for Function Optimization," in *Genetic Algorithms and Soft Computing*, F. Herrera and J. L. Verdegay, Eds.: Physica-Verlag, pp. 1-29, 1995.

[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, 4598, pp. 671-680, 1983.

[14] E. H. L. Aarts and J. Korst, *Simulated Annealing and Boltzman Machines: A Stochastic Approach to Combinatorial and Neural Computing*. New York: John Willey and Sons, 1989.

[15] E. H. L. Aarts, J. H. M. Korst, and P. J. M. Van Laarhoven, "Simulated Annealing," in *Local Search in Combinatorial Optimization*, E. Aarts and K. Lenstra, Eds. New York: John Wiley and Sons, pp. 91-120, 1997.

[16] F. Glover, "Heuristics for Integer Programming using Surrogate Constraints," *Decision Sciences*, vol. 8, pp. 156-166, 1977.

[17] F. W. Glover and M. Laguna, *Tabu Search*: Kluwer Academic Publishers, 1997.

[18] L. J. Fogel, "A Retrospective View and Outlook on Evolutionary Algorithms," in *Computational Intelligence: Theory and Applications, 5th Fuzzy Days*, B. Reusch, Ed. Berlin: Springer-Verlag, pp. 337-342, 1997.

[19] P. Field, "A Multary Theory for Genetic Algorithms: Unifying Binary and Nonbinary Problem Representations," *Ph.D. Thesis, Department of Computer Science*. London: University of London, 1995.

[20] A. J. F. Van Rooji, L. C. Jain, and R. P. Johnson, *Neural Networks Training Using Genetic Algorithms*. New York: World Scientific, 1996.

[21] M. D. Vose, *The Simple Genetic Algorithm: Foundations and Theory*. Cambridge, Massachussets: MIT Press, 1999.

[22] D. B. Fogel, *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. Piscataway, New Jersey: IEEE Pres, 1995.

[23] D. Dasgupta and Z. Michalewicz, "Evolutionary Algorithms: An Overview," in *Evolutionary Algorithms in Engineering Applications*, D. Dasgupta and Z. Michalewicz, Eds. New York: Springer, pp. 3-28, 1997.

[24] J. R. Koza, "Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems," Stanford University Computer Science Department STAN-CS-90-1314, June 1990.

[25] D. B. Fogel, "The Advantages of Evolutionary Computation," in *Bio-Computing and Emergent Computation*, D. Lundh, B. Olsson, and A. Narayanan, Eds. Singapore: World Scientific Press, pp. 1-11, 1997.

[26] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Search," The Santa Fe Institute SFI-TR-95-02-010, 1995.

[27] H. P. Schwefel, "Advantages (and disadvantages) of evolutionary computation over other approaches," in *Evolutionary Computation 1. Basic Algorithms and Operators*, T. Back, D. B. Fogel, and M. Michalewicz, Eds. Philadelphia: Institute of Physics Publishing, pp. 20-22, 2000.

[28] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," *Operations Research*, vol. 21, 2, pp. 498-516, 1973.

[29] L. Kucera, *Combinatorial Algorithms*. Philadelphia: Adam Hilger, 1990.

[30] D. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: a Case Study," in *Local Search in Combinatorial Optimization*, E. Aarts and K. Lenstra, Eds. New York: John Wiley and Sons, pp. 215-310, 1997.

[31] E. H. L. Aarts and H. P. Stehouwer, "Neural Networks and the Traveling Salesman Problem," presented at Proceedings of the International Conference on Artificial Neural Networks, S. Gielen and B. Kappan Eds., Springer, pp. 950-955, Berlin, 1993.

[32] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, Michigan: University of Michigan Press, 1975.

[33] M. Melanie, *An Introduction to Genetic Algorithms*. Cambridge, Massachusetts: The MIT Press, 1996.

[34] D. E. Goldberg, "Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking," *Complex Systems*, 5, pp. 139-167, 1991.

[35] N. N. Schraudolph and R. K. Belew, "Dynamic Parameter Encoding for Genetic Algorithms," *Machine Learning*, vol. 9, pp. 9-21, 1992.

[36] S. Rana and D. Whitley, "Bit Representations with a Twist," presented at the International Conference on Genetic Algorithms, T. Baeck Eds., Morgan Kaufmann, 1997.

[37] S. Rana and D. Whitley, "Search, Binary Representations, and Counting Optima," presented at the Workshop on Evolutionary Algorithms, Springer, vol. 111, pp. 177-190, New York, 1999.

[38] D. Whitley and S. Rana, "Representation, Search, and Genetic Algorithms," presented at the 14th National Conference on Artificial Intelligence (AAAI-97), AAAI Press/MIT Press, 1997.

[39] A. H. Wright, "Genetic Algorithms for Real Parameter Optimization," in *Foundations of Genetics Algorithms*, G. J. E. Rawlins, Ed. San Mateo, California: Morgan Kaufman, pp. 205-221, 1991.

[40] D. Whitley, Starkweather, and D. Shaner, "Scheduling Problems and Traveling Salesman: Genetic Edge Recombination Operator," presented at the Third International Conference on Genetic Algorithms, J. Schaffer Eds., Morgan Kauffman Publishers, pp. 133-140, Los Altos, CA, 1989.

[41] L. J. Eshelman, "Genetic Algorithms," in *Evolutionary Computation 1. Basic Algorithms and Operators*, T. Back, B. D. Fogel, and T. Michalewicz, Eds. New York: Institute of Physics Publishing, pp. 64-80, 2000.

[42] A. E. Eiben, "Multi-parent Recombination," in *Handbook of Evolutionary Computation*, T. Bäck, D. Fogel, and M. Michalewicz, Eds.: IOP Publishing Ltd. and Oxford University Press, 1998.

[43] A. E. Eiben, P. E. Raué, and Z. S. Ruttkay, "Genetic Algorithms with Multi-parent Recombination," presented at the 3rd Conference on Parallel Problem Solving from Nature, Y. Davidor, H.-P. Schwefel, and R. Männer Eds., pp. 78-87, 1994.

[44] A. E. Eiben, C. H. M. Van Kemenade, and K. J.N., "Orgy in the Computer: Multi-parent Reproduction in Genetic Algorithms," presented at the 3rd European Conference on Artificial Life, F. Moran, A. Moreno, J. J. Merelo, and P. Chacon Eds., Springer-Verlag, pp. 934-945, 1995.

[45] A. E. Eiben and V. K. C. H. M., "Diagonal Crossover in Genetic Algorithms for Numerical Optimization," *Journal of Control and Cybernetics*, vol. 26, 3, pp. 447-465, 1997.

[46] J. Lis and A. E. Eiben, "A Multisexual Genetic Algorithm for Multicriteria Optimization," presented at the 4th IEEE Conference on Evolutionary Computation, pp. 59-64, 1997.

[47] D. A. Van Veldhuizen and G. B. Lamont, "Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art," *Evolutionary Computation*, vol. 8, 2, pp. 125-147, 2000.

[48] K. Deb, "Introduction to Selection," in *Evolutionary Computation 1: Basic Algorithms and Operators*, T. Back, B. D. Fogel, and M. Michalewicz, Eds. Bristol, United Kingdom: Institute of Physics Publishing, pp. 166-171, 2000.

[49] A. Brindle, "Genetic Algorithm for Function Optimization," *Ph.D. Thesis, Department of Computing Science*. Edmonton: University of Alberta, 1981.

[50] D. E. Goldberg, K. Deb, and B. Korb, "Do not Wory, Be Messy," presented at the Fourth International Conference on Genetic Algorithms, R. K. Belew and L. Booker Eds., Morgan Kaufmann Publishers, pp. 24-30, San Mateo, CA, 1991.

[51] K. A. De Jong, "Analysis of the Behavior of a Class of Genetic Adaptive Systems," *Ph.D. Thesis, Department of Computer and Communication Sciences*: University of Michigan, 1975.

[52] J. Baker, "Adaptive Selection Methods for Genetic Algorithms," presented at The First International Conference on Genetic Algorithms and Their Applications, J. J. Grefenstette Eds., Lawrence Erlbaum Associates (Hillsdale), pp. 101-111, 1985.

[53] D. Whitley, "The GENITOR Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best," presented at the Third International Conference on Genetic Algorithms, J. D. Schaffer Eds., Morgan Kaufmann., pp. 116-121, San Mateo, CA., 1989.

[54] L. R. Karg and G. L. Thompson, "A Heuristic Approach to Solving Travelling Salesman Problems," *Management Science*, vol. 10, 2, pp. 225-248, 1964.

[55] D. J. Rozenkrantz, R. E. Stearns, and P. M. Lewis, "An Analysis of Several Heuristics for the Traveling Salesman Problem," *SIAM Journal on Computing*, vol. 6, pp. 563-581, 1977.

[56] N. J. Radcliffe and P. D. Surry, "Formal Memetic Algorithms," in *Lecture Notes in Computer Science*. New York: Springer-Verlag, pp. 1-16, 1994.

[57] P. Merz and B. Freisleben, "Genetic Local Search for the TSP: New Results," presented at International Conference on Evolutionary Computation (ICEC 97 April 13-16), N. F. F. Ebecken Eds., Wit Press/Computational Mechanics Publications, Boston, 1997.

[58] F. Glover, "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Operations Research*, vol. 13, 5, pp. 533-549, 1986.

[59] M. Zachariasen and M. Dam, "Tabu Search on the Geometric Traveling Salesman Problem," presented at Metaheuristics International Conference, I. H. Osman and J. P. Kelly Eds., pp. 571-587, Colorado, 1995.

[60] O. C. Martin and S. W. Otto, "Combining Simulated Annealing with Local Search Heuristics," *Annals of Operations Research*, vol. 63, pp. 57-75, 1996.

[61] J. J. Hopfield and D. W. Thank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, 52, pp. 141-152, 1985.

[62] T. Kohonen, "The Self-organizing Map," *IEEE*, vol. 78, 9, pp. 1464-1480, 1990.

[63] B. Fritzke and P. Wilke, "A Neural Network For the Traveling Salesman Problem With Linear Time And Space Complexity," presented at the Internatioanal Joint Conference on Neural Networks (IJCNN), IEEE Service Center, pp. 929-934, Singapore, 1991.

[64] M. Dorigo and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Transaction on Evolutionary Computation*, vol. 1, 1, 1997.

[65] M. Dorigo and L. M. Gambardella, "Ant Colonies for the Traveling Salesman Problem," *BioSystems*, vol. 43, pp. 73-81, 1997.

[66] P. Jog, J. Y. Suh, and D. V. Gucht, "Parallel Genetic Algorithms Applied to the Traveling Salesman Problem," *SIAM Journal on Optimization*, vol. 1, 4, pp. 515-529, 1991.

[67] K. F. Pal, "Genetic Algorithms for the Traveling Salesman Problem Based on a Heuristic Crossover Operation," *Biological Cybernetics*, 69, pp. 539-546, 1993.

[68] C. L. Valenzuela and L. P. Williams, "Improving Simple Heuristic Algorithms for the Travelling Salesman Problem using a Genetic Algoritm," presented at the Seventh International Conference on Genetic Algortihms, pp. 458-464, 1997.

[69] W. Lin, J. G. Delgado-frias, D. C. Gause, and S. Vassiliadis, "Hybrid Newton-Raphson Genetic Algorithm for the Traveling Salesman Problem," *Cybernetics and Systems: An Internatonal Journal*, vol. 26, pp. 387-412, 1995.

[70] S. J. Louis and G. Li, "Case Injection Genetic Algorithms for Traveling Salesman Problems," *Information Sciences*, pp. 201-225, 2000.

[71] J. P. Watson, C. Ross, V. Eisele, J. Bins, C. Guerra, L. D. Whitley, and A. Howe, "The Traveling Salesrep Problem: Edge Assembly Crossover," presented at Parallel Problem Solving from Nature V, A. E. Eiben Eds., Springer-Verlag, 1998.

[72] V. M. Kureichick, V. V. Miagkikh, and A. P. Topchy, "Genetic Algorithm for Solution of the Traveling Salesman Problem with New Features against Premature Convergence," presented at ECDC - 96, Plymouth, UK, 1996.

[73] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A Study of Permutation Crossover Operators on the TSP," presented at the Second International Conference on Genetic Algorithms and Their Applications, J. J. Grefenstette Eds., Lawrence Erlbaum, pp. 224-230, Hilldale, New Jersey, 1987.

[74] S. S. Lam, K. W. C. Tang, and X. Cai, "Genetic Algorithm with Pigean-hole Coding Scheme for Solving Sequencing Problems," *Applied Artificial Intellingence*, vol. 10, pp. 239-256, 1996.

[75] M. Gorges-Schleuter, "Asparagos96 and the Traveling Salesman Problem," presented at IEEE International Conference on Evolutionary Computation, pp. 171-174, 1997.

[76] Y. Nagata and S. Kobayashi, "Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem," presented at the ICGA'97, pp. 450-457, 1997.

[77] J. Bean, "Genetic Algorithms and Random Keys for Sequencing and Optimization," *ORSA Journal on Computing*, vol. 6, 2, pp. 154-160, 1994.

[78] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*. New York: John Willey and Son, Inc., 1997.

[79] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains," presented at the International Joint Conference on Artificial Intelligence, D. Dasgupta and Z. Michalewicz Eds., Springer, pp. 162-164, New York, 1985.

[80] D. Whitley, Starkweather, and D. Shaner, "The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination," in *Handbook of Genetic Algorithms*, L. Davis, Ed. New York: Van Nostrand Reinhold, pp. 350-376, 1991.

[81] B. R. Fox and M. B. McMahon, "Genetic Operators for Sequencing Problems," in *Foundations of Genetics Algorithms*, G. J. E. Rawlins, Ed. San Mateo, California: Morgan Kaufman, pp. 284-300, 1991.

[82] D. R. Frantz, "Non-linearities in Genetic Adaptive Search," *Ph.D. Thesis, Department of Computer and Communication Sciences*: University of Michigan, 1972.

[83] S. Chatterjee, C. Carrera, and L. A. Lynch, "Genetic Algorithms and Traveling Salesman Problems," *European Journal of Operational Research*, vol. 93, pp. 439-450, 1996.

[84] TSPLIB, "http://softlib.rice.edu/softlib/tsplib/,"., 1995.

[85] L. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*. New York: Wiley Publishing, 1966.

[86] B. D. Fogel, "Applying Evolutionary Programming to Selected Traveling Salesman Proplem," *Cybernetics and Systems: An Internatonal Journal*, vol. 24, pp. 27-36, 1993.

[87] D. B. Fogel, "An Evolutionary Approach to the Traveling Salesman Problem," *Biological Cybernetics*, vol. 60, 2, pp. 139-144, 1988.

[88] S. Eilon, C. D. T. Watson-Gandy, and N. Christofides, "Distribution Management: Mathematical Modeling and Practal Analysis," *Operational Research Quarterly*, vol. 20, pp. 37-53, 1969.

[89] C. Guerra-Salcedo and D. Whitley, "Genetic Search for Feature Subset Selection: A Comparison Between CHC and GENESIS," *Proceedings of the Symposium on Genetic Algorithms*, 1998.

[90] S. Choenni, "On the Suitability of Genetic-based Algorithms for Data Mining," Nationaal Lucht- en Ruimtevaarlabaratorium National Aerospace Laboratory NLR NLR-TP-98484, November 1998.

[91] C. J. Meneses and G. G. Grinstein, "Categorization and Evaluation of Data Mining Techniques," presented at International Conference on Data Mining, N. F. F. Ebecken Eds., Wit Press/Computational Mechanics Publications, pp. 54-69, Boston, 1998.

[92] R. Kewley, M. J. Embrechts, and C. M. Breneman, "Neural Network Analysis for Data Strip Mining Problems," in *Intelligent Engineering Systems through Artificial Neural Networks*, vol. 8, C. Dagli, Ed. Nashville - Missouri: ASME Press, pp. 391-396, 1998.

[93] C. Emmanouilidis, A. Hunter, and J. MacIntyre, "A Multi Objective Evolutionary Setting for Feature Selection and a Commonality-Based Crossover Operator," presented at 2000 Congress on Evolutionary Computation, IEEE Service Center, vol. 1, pp. 309-316, Piscataway, New Jersey., 2000.

[94] J. H. Friedman, "On Bias, Variance, 0/1-Loss, and the Curse-of-Dimensionality," *Data Mining and Knowledge Discovery.*, vol. 1, pp. 55-77, 1997.

[95] K. Deng and A. Moore, "On the Greediness of Feature Selection Algorithms," presented at International Conference of Machine Learning (ICML '98), H. J. Van den Herik and T. Weijters Eds., Universiteit Maastricht, The Netherlands, 1998.

[96] J. Devillers, "Genetic Algorithms in Computer-aided Molecular Design," in *Genetics Algorithms in Molecular Modeling*, J. Devillers, Ed. San Diego: Harcourt Brace & Company, pp. 1-35, 1996.

[97] F. Arciniegas, K. P. Bennett, C. M. Breneman, and M. J. Embrechts, "Molecular Database Mining using Self-Organizing Maps for the Design of Novel Pharmaceuticals," presented at Intelligent Engineering Systems Through Artificial Neural Networks (ANNIE) Conference in Cooperation with the IEEE Neural Network Council, C. Dagli Eds., ASME Press, vol. 10, pp. 477-482, St. Louis, Missouri, 2000.

[98] G. Schneider, "Neural Networks are Useful Tools for Drug Design," *Neural Networks*, 13, pp. 15-16, 2000.

[99] W. J. Dunn and D. Rogers, "Genetic Partial Least Squares in QSAR," in *Genetic Algorithms in Molecular Modeling*, J. Devillers, Ed. New York: Academic Press, pp. 109-129, 1996.

[100] W. Zheng and A. Tropsha, "Novel Variable Selection Quantitative Structure - Property Relationship Approach Based on the k-Nearest-Neighbor Principle," *Journal of Chemical Information and Computer Sciences*, vol. 40, pp. 185-194, 2000.

[101] H. Kubinyu, "Variable Selection in QSAR Studies. I. An Evolutionary Algorithm," *Quant. Struct.-Act. Relat.*, vol. 13, pp. 285-294, 1994.

[102] H. Kubinyu, "Variable Selection in QSAR Studies. II. A Highly Efficient Combination of Systematic Search and Evolution," *Quant. Struct.-Act. Relat.*, vol. 13, pp. 393-401, 1994.

[103] H. Kubinyu, "Evolutionary Variable Selection in Regression and PLS Analyses," *Journal of Chemometrics*, vol. 10, pp. 110-133, 1996.

[104] P. Geladi and B. Kowalski, "Partial Least Squares Regression: A Tutorial," *Chimica Acta*, vol. 185, pp. 1-17, 1986.

[105] M. A. Sharaf, D. L. Ilman, and B. Kowalski, *Chemometrics*. New York: John Willey & Sons, 1986.

[106] M. Baroni, G. Costantino, G. Cruciani, D. Riganelli, R. Valigi, and S. Clementi, "Generating Optimal Linear PLS Estimations (GOLPE): An Advanced Chemometric Tool for Handling 3D-QSAR Problems," *Quantitative Structure-Activity Relationship*, vol. 12, pp. 9-20, 1993.

[107] S. Wold, E. Johansson, and M. Cocchi, in *3D QSAR in Drug Design: Theory, Methods, and Applications*, H. Kubinyu, Ed. Leiden: ESCOM, pp. 523-550, 1993.

[108] F. Lindgren, P. Geladi, S. Rannar, and S. Wold, "Interactive Variable Selection (IVS) for PLS. Part 1: Theory and Algorithms," *Journal of Chemometrics*, vol. 8, 5, pp. 349-363, 1994.

[109] G. Cruciani, S. Clementi, and M. Baroni, in *3D QSAR in Drug Design: Thoeory, Methods, and Applications*, H. Kubinyu, Ed. Leiden: ESCOM, pp. 551-564, 1993.

[110] S. Rännar, F. Lindgren, P. Geladi, and S. Wold, "A PLS Kernel Algorithm for Data Sets with Many Variables and Fewer Objects. Part 1: Theory and Algorithm," *Journal of Chemometrics*, vol. 8, 2, pp. 111-126, 1996.

[111] N. Kettanehwold, J. F. Macgregor, B. Dayal, and S. Wold, "Multivariate Design of Process Experiments (M-Dope)," *Chemometrics and Intelligent Laboratory Systems*, vol. 23, pp. 39-50, 1994.

[112] F. Lindgren, P. Geladi, A. Berglund, M. Sjöström, and S. Wold, "Interactive Variable Selection (IVS) for PLS. Part II: Chemical Applications," *Journal of Chemometrics*, vol. 9, 5, pp. 331-342, 1995.

[113] A. Ajay, "A Unified Framework for Using Neural Networks to Build QSARs," *Journal Medicinal Chemistry*, vol. 36, pp. 3565-3571, 1993.

[114] D. T. Manallack, D. D. Ellis, and D. J. Livingstone, "Analysis of Linear and Nonlinear QSAR Data Using Neural Networks," *Journal Medicinal Chemistry*, vol. 37, pp. 3758-3767, 1994.

[115] T. J. Hou, J. M. Wang, N. Liao, and X. J. Xu, "Applications of Genetic Algorithms on the Structure-Activity Relationship Analysis of Some Cinnamamides," *Journal of Chemical Information and Computer Sciences*, vol. 39, pp. 775-781, 1999.

[116] D. Liu, H. Jiang, K. Chen, and R. Ji, "A New Approach to Design of Combinatorial Library with Genetic Algorithm Based on 3D Grid Property," *Journal of Chemical Information and Computer Sciences*, vol. 38, pp. 233-242, 1998.

[117] H. Chen, J. Zhou, and G. Xie, "PARM: A Genetic Evolved Algorithm to Predict Bioactivity," *Journal of Chemical Information and Computer Sciences*, vol. 38, pp. 243-250, 1998.

[118] B. T. Hoffman, T. Kopajtic, L. Katz, and A. H. Newman, "2D QSAR Modeling and Preliminary Database Searching for Dopamine Transporter Inhibitors Using Genetic Algorithm Variable Selection of Moconn Z Descriptors," *Journal of Chemical Information and Computer Sciences*, vol. 43, pp. 4151-4159, 2000.

[119] T. Kimura, K. Hasegawa, and K. Funatsu, "GA Strategy for Variable Selection in QSAR Studies: GA-Based Region Selection for CoMFA Modeling," *Journal of Chemical Information and Computer Sciences*, vol. 38, pp. 276-282, 1998.

[120] K. Hasegawa, T. Kimura, and K. Funatsu, "GA Strategy for Variable Selection in QSAR Studies: Application of GA-based Region Selection to a 3D-QSAR Study of Acetylcholinesterase," *Journal of Chemical Information and Computer Sciences*, vol. 39, pp. 112-120, 1999.

[121] B. T. Luke, "Evolutionary Programming Applied to the Development of Quantitative Structure-Activity Relationships and Quantitative Structure-Property Relationships," *Journal of Chemical Information and Computer Sciences*, vol. 34, pp. 1279-1287, 1994.

[122] C. L. Waller and M. P. Bradley, "Development and Validation of a Novel Variable Selection Technique with Application to Multidimensional Quantitative Structure- Activity Relationship Studies," *Journal of Chemical Information and Computer Sciences*, vol. 39, 345-355, 1999.

[123] M. D. Wessel, P. C. Jurs, J. W. Tolan, and S. M. Muskal, "Prediction of Human Intestinal Absorption of Drug Compounds from Molecular Structure," *Journal of Chemical Information and Computer Sciences*, vol. 38, pp. 726-735, 1998.

[124] S. R. Johnson and P. C. Jurs, "Prediction of Acute Mammalian Toxicity from Molecular Structure for a Diverse Set of Substituted Anilines Using Regression Analysis and Computational Neural Networks," in *Computer-Assisted Lead Finding and Optimization*, H. Van De Waterbeemd, B. Testa, and G. Folkers, Eds. New York: Wiley-VCH, pp. 29-48, 1997.

[125] C. L. Mallows, "Some Comments on Cp," *Technometric*, vol. 15, pp. 661-675, 1973.

[126] S. Chatterjee, A. S. Hadi, and B. Price, *Regression Analysis by Example*, Third ed. New York: Wiley Series, 2000.

[127] M. L. Thompson, "Selection of Variables in Multiple Regression. Part 1. A Review and Evaluation," *International Statistical Review*, vol. 46, pp. 1-19, 1978.

[128] M. L. Thompson, "Selection of Variables in Multiple Regression. Part 2. Chosen Procedures, Computations and Examples," *International Statistical Review*, vol. 46, pp. 129-146, 1978.

[129] H. Akaike, "A New Look at the Statistical Identification Model," *IEEE Transactions on Automatic Control*, vol. 19, pp. 716-723, 1974.

[130] G. Schwarz, "Estimating the Dimension of a Model," *Annals of Statistics*, vol. 6, pp. 461-464, 1978.

[131] P. Leray and P. Gallinari, "Feature Selection with Neural Networks," the Laboratoire d'Informatique de Paris 6. Université Pierre et Marie Curie LIP6 TR 1998-012, 1998.

[132] D. Skalak, "Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms," presented at Eleventh International Machine Learning Conference, Morgan Kaufmann, pp. 293-301, New Brunswick, NJ, 1994.

[133] H. Vafaie and K. De Jong, "Genetic Algorithms as a Tool for Feature Selection in Machine Learning," presented at the 4th International Conference on Tools with Artificial Intelligence, pp. 200-204, Arlington, VA., 1992.

[134] J. J. Grefenstette, "GENESIS: A System for Using Genetic Search Procedures," presented at the Conference on Intelligent Systems and Machines, pp. 161-165, 1984.

[135] L. J. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination," in *Foundations of Genetic Algorithms*: Morgan Kaufmann, pp. 265-283, 1991.

[136] A. Yasri and D. Hartsough, "Toward an Optimal Procedure for Variable Selection and QSAR Model Building," *Journal of Chemical Information and Computer Sciences*, vol. 41, pp. 1218-1227, 2001.

[137] V. Vapnik, *The Nature of Statistical Learning*. New York: Springer, 1995.

[138] F. Z. Brill, D. E. Brown, and W. N. Martin, "Fast Genetic Selection of Features for Neural Networks Classifiers," *IEEE Transactions on Neural Networks*, vol. 3, pp. 324-328, 1992.

[139] F. M. Ham and I. Kostanic, *Principles of Neurocomputing for Science and Engineering*. New York: McGraw Hill, 2001.

[140] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, New Jersey: Prentice-Hall, Inc., 1999.

[141] P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences,".: Harward, 1974.

[142] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally Weighted Learning," *Artificial Intelligence Review*, vol. 11, pp. 11-73, 1997.

[143] L. Bottou and V. N. Vapnik, "Local Learning Algorithms," *Neural Computation*, vol. 4, 6, pp. 888-900, 1992.

[144] M. J. Embrechts, D. Devogelaere, and M. Rijckaert, "Supervised Scaled Regression Clustering: an Alternative to Neural Networks," presented at IEEE-INNS-ENNS International Conference (IJCNN) (July, 24-27), vol. 6, pp. 571-576, Como. Italy, 2000.

[145] A. J. Smola and B. Scholkopf, "A Tutorial on Support Vector Regression," NeuroCOLT2 Technical Report NC2-TR-1998-030, 1998.

[146] M. Momma and K. P. Bennett, "A Pattern Search Method for Model Selection of Support Vector Regression," presented at SIAM International Conference on Data Mining, Arlington, VA, 2002.

[147] K. P. Bennett, A. Demiriz, C. M. Breneman, and M. J. Embrechts, "Support Vector Machine Regression in Chemometrics," presented at 33rd Symposium on the Interface of Computing Science and Statistics, Interface'01: Frontiers in Data Mining and Bioinformatics, Costa Mesa - CA, 1991.

[148] R. Collobert and S. Bengio, "Support Vector Machines for Large-Scale Regression Problems,", IDIAP-RR-00-17, 2000.

[149] J. E. Dennis and V. J. Torczon, "Derivative-Free Pattern Search Methods for Multidisciplinary Design Problems," presented at the 5th AIAA/ USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization (Sept. 7-9), pp. 922-932, Panama City, FL, 1994.

[150] O. Chapelle and V. N. Vapnik, "Model Selection for Support Vector Machines," presented at Advances in Neural Information Processing Systems, S. A. Solla, T. K. Leen, and M. K. R. Eds., pp. 230-236, Cambridge, MA, 2000.

[151] S. Wold, A. Ruhe, H. Wold, and W. J. Dunn III, "The Collinearity Problem in Linear Regression. The Partial Least Squares (PLS) Approach to Generalized Inverses," *SIAM Journal of Scientific Statistical Computing*, vol. 5, 3, pp. 735-743, 1984.

[152] D. Livingstone, *Data Analysis for Chemists*. Oxford: Oxford Science Publications, 1995.

[153] J. M. Zurada, A. Malinowski, and A. Cloete, "Sensitivity Analysis for Minimization of Input Dimension for Feedforward Neural Networks," presented at IEEE International Symposium on Circuits and Systems (May 30 -June 3), vol. 6, pp. 447-450, London, 1994.

[154] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," presented at Advances in Neural Information Processing Systems, D. S. Touretzky Eds., Morgan Kaufmann, vol. 2, pp. 598-605, San Mateo, CA., 1990.

[155] F. Arciniegas, "Feature Selection and Statistical Alternatives for Machine Learning Applied to In-Silico Drug Design," *Unpublished Ph.D. Thesis, Decision Sciences and Engineering Systems*. Troy, New York: Rensselaer Polytechnic Institute, 2002.

[156] B. Efron, *The Jackknife, the Bootstrap, and Other Resampling Plans*. Philadelphia, PA., 1982.

[157] B. Efron and R. Tibshirani, "Cross-validation and the Bootstrap: Estimating the Error Rate of a Prediction Rule," Department of Statistics, Stanford University 1995.

[158] C. M. Breneman, T. R. Thompson, M. Rhem, and M. Dung, "Electron Density Modeling of Large Systems Using the Transferable Atom Equivalent Method," *Computers & Chemistry*, vol. 19, 3, pp. 161-179, 1995.

[159] C. M. Breneman and M. Rhem, "A QSPR Analysis of HPLC Column Capacity Factors for a Set of High-Energy Materials Using Electronic Van der Waals Surface Property Descriptors Computed by the Transferable Atom Equivalent Method," *Journal of Computational Chemistry*, vol. 18, 2, pp. 182-197, 1997.

[160] R. J. Zauhar and W. J. Welsh, "Application of the "Shape Signatures" Approach to Ligand- and Receptor-based Drug Design," presented at American Chemical Society 220, Washington D.C., 2000.

[161] C. M. Sundling and C. M. Breneman, "Advances in Electronic Property-encoded Molecular Shape Descriptors," presented at American Chemical Society (April, 2001) Poster presentation, San Diego, 2001.

[162]. Chemical Computing Group, http://www.chemcomp.com/.

[163] C. M. Breneman, N. Sukumar, K. P. Bennett, M. J. Embrechts, M. Sundling, and L. Lockwood, "Wavelet Representations of Molecular Electronic Properties: Applications in ADME, QSPR, and QSAR," presented at ACS National Meeting, Washington D.C., 2000.

[164] F. Lombardo, J. F. Blake, and W. J. Curatolo, "Computation of Brain-Blood Partitioning of Organic Solutes via Free Energy Calculations," *Journal Medicinal Chemistry*, vol. 39, pp. 4750-4755, 1996.

[165] R. Garg, S. P. Gupta, H. Gao, M. S. Babu, A. K. Debnath, and C. Hansch, "Comparitive QSAR Studies on Anti-HIV Drug," *Chemistry Reviews*, vol. 99, pp. 3525-3601, 1999.

[166] H. J. Breslin, M. J. Kukla, D. W. Ludovici, R. Mohrbacher, W. Ho, M. Miranda, J. D. Rodgers, T. K. Hitchens, G. Leo, D. A. Gauthier, C. Y. Ho, M.K. Scott, E. D. Clerq, R. Pauwels, K. Andries, M. A. C. Janssen, and P. A. J. Janssen, "Synthesis and Anti-HIV-1 Activity of 4,5,6,7-Tetrahydro-5-methylimidazo-[4,5,1-jk]benzodiazepin-2(1H)-one (TIBO) Derivatives," *Journal Medicinal Chemistry*, vol. 37, pp. 771-793, 1995.

[167] H. Tanaka, H. Takashima, M. Ubasawa, K. Sekiya, I. Nitta, M. Baba, S. Shigeta, R. T. Walker, E. D. Clerq, and T. Miyasaka, "Structure-Activity Relationships of 1-[(2-hydroxyethoxy)methyl]-6-(phenythio)thymine Analogues: Effect of Substitutions at the C-6 Phenyl Ring and at the C-5 Position on Anti-HIV-1 Activity," *Journal Medicinal Chemistry*, vol. 35, pp. 337-345, 1992.

[168] A. San-Felix, V. Sonsoles, M.-J. Perez-Perez, J. Balzarini, E. De Clerq, and M. J. Camarasa, "Novel Series of TSAO-T Derivatives, Synthesis and Anti-HIV-1 Activity of 4-, 5-, and 6-Substituted Pyrimidine Analogs," *Journal Medicinal Chemistry*, vol. 37, pp. 453-460, 1994.

[169] R. Alvarez, S. Velazquez, A. San-Felix, S. Aquaro, E. De Clerq, C.-F. Perno, A. Karlsson, J. Balzarini, and M. J. Camarasa, "1,2,3-Triazole-[2,5-Bis-O-(tert-butyldimethylsilyl)-.beta.-D-ribofuranosyl]-3'-spiro-5"-(4"-amino-1",2"-oxathiole 2",2"-dioxide) (TSAO) Analogs: Synthesis and Anti-HIV-1 Activity," *Journal Medicinal Chemistry*, vol. 37, pp. 4185-4194, 1994.

[170] Y. Hanasaki, H. Watanabe, K. Katsuura, H. Takayama, S. Shirakawa, K. Yamaguchi, S.-I. Sakai, K. Ijichi, and M. Fujiwara, "Thiadiazole Derivatives: Highly Potent and Specific HIV-1 Reverse Transcriptase Inhibitors," *Journal Medicinal Chemistry*, vol. 38, pp. 2038-2040, 1995.

[171] A. K. Debnath, "Three Dimensional QSAR Study on Cyclic Urea Dervatives as HIV-1 Protease Inhibitors: Application of COMFA," *Journal Medicinal Chemistry*, vol. 42, pp. 249-259, 1999.

[172] J. Fogh and G. Trempe, "New Human Tumor Cell Lines," in *Human Tumor Cells in Vitro*, J. Fogh, Ed. New York: Plenum, pp. 115-141, 1975.

[173] J. Fogh and T. Orfeo, "One Hundred and Twenty-seven Cultured Human Tumor Cell Lines Producing Tumors in Nude Mice," *Journal of the National Cancer Institute*, vol. 59, pp. 221-225, 1977.

[174] P. Stenberg, K. Luthman, and P. Artursson, "Virtual Screening of Intestinal Drug Permeability," *Journal of Controlled Release*, vol. 65, 1-2, pp. 231-243, 2000.

[175] P. Stenberg, U. Nornider, K. Luthman, and P. Artursson, "Experimental and Computational Screening Models for the Prediction of Intestinal Drug Absorption," *Journal of Medicinal Chemistry*, vol. 44, 12, pp. 1927-1937, 2001.

[176] M. Yazdanian, S. L. Glynn, J. L. Wright, and A. Hawi, "Correlating Partitioning and Caco-2 Cell Permeability of Structurally Diverse Small Molecular Weight Compounds," *Pharm. Res.*, vol. 9, 15, pp. 1490-1494, 1998.

[177] W. A. Spears, K. A. De Jong, T. Baeck, D. B. Fogel, and H. De Garis, "An Overview of Evolutionary Computation," presented at European Conference on Machine Learning, P. V. Brazdil Eds., Springer Verlag, vol. 667, pp. 442-459, 1993.

[178] M. A. Hall, "Correlation-based Feature Selection for Machine Learning," *Ph.D. Thesis, Department of Computer Science*. Hamilton, New Zeland: The University of Waikato, 1999.

[179] G. Keller, B. Warrack, and H. Bartel, *Statistics for Management and Economics*, 3 ed. Belmont, California: Duxbury Press, 1994.

[180] S. Wold, "PLS for Multivariate Linear Modeling," in *Chemometric Methods in Molecular Design*, H. Van De Waterbeemd, Ed. Weinheim: VCH Publishers, Inc., pp. 197-218, 1995.

[181] S. Wold, "PLS-regression: A Basic Tool of Chemometrics," *Chemometrics and Intelligent Laboratory Systems*, vol. 58, pp. 109-130, 2001.

[182] S. Wold, N. Kettaneh, and K. Tjessem, "Hierarchical Multiblock PLS and PC Models for Easier Model Interpretation and as an Alternative to Variable Selection," *Journal of Chemometrics*, vol. 10, pp. 463-482, 1996.

[183] R. Leardi and A. L. Gonzales, "Genetic Algorithms Applied to Feature Selection in PLS Regression: How and When to Use Them," *Chemometrics and Intelligent Laboratory Systems*, vol. 41, pp. 195-207, 1998.

[184] H. Martens and T. Naes, *Multivariate Calibration*. New York: John Wiley & Sons, 1989.

[185] M. C. Dehman, "Implementing Partial Least Squares," *Statistics and Computing*, vol. 5, pp. 191-202, 1995.

[186] P. Geladi and B. Kowalski, "An Example of 2-Block Predictive Partial Least - Squares Regression with Simulated Data," *Analytica Chimica Acta*, vol. 185, pp. 19-32, 1986.

[187] H. A. Martens and P. Dardenne, "Validation and Verification of Regression in Small Data Sets," *Chemometrics and Intelligent Laboratory Systems*, vol. 44, pp. 99-121, 1998.

[188] S. Wold, J. Trygg, A. Berglund, and H. Antti, "Some Recent Development in PLS Modeling," *Chemometrics and Intelligent Laboratory Systems*, vol. 58, pp. 131-150, 2001.

[189] S. Wold, N. Kettaneh, and B. Skagerberg, "Non-linear PLS Modeling," *Chemometrics and Intelligent Laboratory Systems*, vol. 7, pp. 53-65, 1989.

[190] S. Wold, "Non-linear Partially Least Squares Modelling. II Splines Inner Functions," *Chemometrics and Intelligent Laboratory Systems*, vol. 14, pp. 71-84, 1992.

[191] G. Baffi, E. B. Martin, and A. J. Morris, "Non-linear Projection to Latent Structures Revisited: The Quadratic PLS Algorithm," *Computers and Chemical Engineering*, vol. 23, pp. 395-411, 1999.

[192] A. Berglund and S. Wold, "INLR, Implicit Non-linear Latent Variable Regression," *Journal of Chemometrics*, vol. 11, pp. 141-156, 1997.

[193] T. R. Holcomb and M. Morari, "PLS/Neural Networks," *Computers and Chemical Engineering*, vol. 16, 4, pp. 393-411, 1992.

[194] S. J. Qin and T. J. Macovay, "Nonlinear PLS Modeling using Neural Networks," *Computers and Chemical Engineering*, vol. 16, 4, pp. 379-391, 1992.

[195] A. Berglund, N. Kettaneh, L. Uppgard, S. Wold, N. Bendwell, and D. R. Cameron, "The GIFI Approach to Non-linear PLS Modeling," *Journal of Chemometrics*, vol. 15, pp. 321-336, 2001.

[196] R. Leardi, "Genetic Algorithm in Feature Selection," in *Genetic Algorithms in Molecular Modeling*, J. Devillers, Ed. London: Academic Press, pp. 67-87, 1996.

[197] K. Hasegawa, Y. Miyashita, and K. Funatsu, "GA Strategy for Variable Selection in QSAR Studies: GA-based PLS Analysis of Calcium Channel Antagonists," *Journal of Chemical Information and Computer Sciences*, vol. 37, pp. 306-310, 1997.

[198] J. Friedman, "Multivariate Adaptive Regression Splines," Laboratory for Computational Statistics, Department of Statistics, Stanford University, Palo Alto, CA 1988.

[199] J. J. Grefenstette, "Rank-based Selection," in *Introduction to Selection, in Evolutionary Computation 1: Basic Algorithms and Operators*, T. Back, B. D.

Fogel, and M. Michalewicz, Eds. Bristol, United Kingdom: Institute of Physics Publishing, pp. 187-194, 2000.

[200] R. Leardi, "Application of a Genetic Algorithm to Feature Selection Under Full Validation Conditions and to Outlier Detection," *Journal of Chemometrics*, vol. 8, pp. 65-79, 1994.

[201] S. Lanteri, "Full Validation Procedures for Feature Selection in Classification and Regression Problem," *Chemometrics and Intelligent Laboratory Systems*, vol. 15, pp. 159-169, 1992.

[202] L. Eriksson, E. Johansson, M. Muller, and S. Wold, "On the Selection of the Training Set in Environmental QSAR Analysis When Compounds are Clustered," *Journal of Chemometrics*, vol. 14, pp. 599-616, 2000.

[203] S. Wold and L. Eriksson, "Statistical Validation of QSAR Results," in *Chemometric Methods in Molecular Design*, H. Van De Waterbeemd, Ed. Weinheim: VCH Publishers, Inc, pp. 309-318, 1995.

[204] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," presented at International Joint Conference on Artificial Intelligence (IJCAI), pp. 1137-1145, 1995.

[205] "Cerius² Relase 4.5 User Manual,"., 2000.

[206] W. W. Hauck and A. Donner, "Wald's Test as Applied to Hypotheses in Logit Analysis," *Journal of the American Statistical Association*, vol. 72, pp. 851-853, 1977.

[207] G. M. Furnival and R. W. J. Wilson, "Regressions by Leaps and Bounds," *Technometrics*, vol. 16, pp. 499-511, 1974.

[208] G. Bontempi, "Local Learning Techniques for Modeling, Prediction, and Control," *Ph.D. Thesis, Applied Sciences*. Brucelles, Belgium: Universite' Libre de Bruxelles, 1999.

[209] M. J. Embrechts, A. Demiriz, and K. P. Bennett, "Supervised Scaled Regression Clustering With Genetic Algorithms," *Intellingent Engineering Systems through Artificial Neural Networks*, vol. 9, pp. 457-462, 1999.

[210] M. L. Raymer, P. C. Sanschagrin, W. F. Punch, S. Venkataraman, E. D. Goodman, and L. A. Kuhn, "Predicting Conserved Water-mediated and Polar Ligand Interactions in Proteins using a K-nearest-neighbors Genetic Algorithm," *Journal of Molecular Biology*, vol. 265, pp. 445-464, 1997.

[211] M. L. Raymer, W. F. Punch, E. D. Goodman, P. C. Sanschagrin, and L. A. Kuhn, "Simultaneous Feature Scaling and Selection using a Genetic Algorithm," presented at the 7 th International Conference on Genetic Algorithms (July, 19-23, 1997), T. Back Eds., Morgan Kaufmann, pp. 561-567, East Lansing, MI., 1997.

[212] G. Demiroz and H. Guvenir, "Genetic Algorithms to Learn Feature Weights for the Nearest Neighbor Algorithm," presented at 6th Belgian-Dutch Conference on Machine Learning (BENELEARN-96) (September 20, 1996), H. J. van den Herik and T. Weijters Eds., pp. 117-126, Universiteit Maastricht, The Netherlands, 1996.

[213] J. Michailidis and J. d. Leeuw, "The GIFI System of Descriptive Multivariate Analysis," *Stat. Sci.*, vol. 13, pp. 307-336, 1998.

[214] D. Steinberg and P. Colla, *CART: A Salford Systems Implementation of the Original Program by Leo Breiman, Jerome Freidman, Richard Olshen, and Charles Stone*: Salford Systems, 1997.

[215] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Pacific Grove: Wadsworth, 1984.

APPENDIX A

The NIPALS Algorithm

X-variables are pre-scaled to ensure that comparable noise level. Then, the average value of each column (variable) of X is calculated and subtracted from corresponding column (mean-centering). The NIPALS algorithm extracts one principal component a time. The NIPALS algorithm proceeds as follows [104], where the symbol ^{T} stands for the transpose of the matrix or the vector:

Step 1. Take a vector x_j from X and call it $t_h : t_h = x_j$

Step 2. Calculate $p_h^T : p_h^T = \frac{t_h^T X}{t_h^T t_h}$

Step 3. Normalize p_h^T to length 1: $p_{(new)h}^T = \frac{p_{(old)h}^T}{\left\| p_{(old)h}^T \right\|}$

Step 4. Calculate $t_h : t_h = \frac{X\ p_h}{p_h^T p_h}$

Step 5. Calculate the residual $E_h = X - t_h p_h^T$

Step 6. Compare the t_h used in the Step 2 with that calculated in Step 4. If both are the same stop (the iteration has converged), else set $X = E_h$ and go to Step 2.

APPENDIX B

The PLS Algorithm

Graphical representation of the dimensions of the matrices and vectors used in PLS algorithm is depicted in the Figure B.1.The symbol ^{T} stands for the transpose of the matrix or the vector. The PLS algorithm proceeds as follows [104]:

Step 0. X and Y are mean-centered and scaled to unit variance.

Step 1. As starting values, take $u_h = Y_{i1}$

Step 2. $w_h^T = \dfrac{u_h^T X}{u_h^T u_h}$

Step 3. w_h^T is normalized to unity: $w_{(new)h}^T = \dfrac{w_{(old)h}^T}{\left\| w_{(old)h}^T \right\|}$

Step 4. $t_h^T = \dfrac{X w_{(new)h}}{w_{(new)h}^T w_{(new)h}}$

Step 5. $q_h^T = \dfrac{t_h^T Y}{t_h^T t_h}$

Step 6. q_h^T is normalized to unity: $q_{(new)h}^T = \dfrac{q_{(old)h}^T}{\left\| q_{(old)h}^T \right\|}$

Step 7. $u_h = \dfrac{Y q_{(new)h}}{q_{(new)h}^T q_{(new)h}}$

Step 8. Perform a convergence check to see if the u used in Step 2 with that calculated in Step 7 are the same or not within a predetermined range. If $\left\| u_{(step2)h} - u_{(step7)h} \right\| < \delta$ where δ is some predefined threshold; then go to Step 9. Else return to Step 2 using the new value of u from Step 7. (If Y has only one column steps 5 through 8 can be omitted by putting q = 1, and no more iteration is necessary.)

Steps 9 through 12 calculate X loadings and rescale the scores and weights accordingly:

Step 9. $p_h^T = \dfrac{t_h^T X}{t_h^T t_h}$

Step 10. p_h^T is normalized to unity: $p_{(new)h}^T = \dfrac{p_{(old)h}^T}{\left\| p_{(old)h}^T \right\|}$

Step 11. $t_{(new)h}^T = t_h^T \left\| p_{(old)h}^T \right\|$

Step 12. $w_{(new)h}^T = w_{(old)h}^T \left\| p_{(old)h}^T \right\|$

Save $p_{(new)h}^T$ $q_{(new)h}^T$ and $w_{(new)h}^T$ for prediction.

Step 13. Find the regression coefficient b_h for inner relationship:

$$b_h = \frac{u^T t_{(new)h}}{t_{(new)h}^T t_{(new)h}}$$

Step 15. Calculate the residual E_h and F_h for the X-block and for the Y-block, respectively.

The general outer relation for the X-block for the component h:

$$E_h = E_{h-1} - t_{(new)h} p^T_{(new)h}\ ;\ X = E_0$$

The mixed relation for the Y-block for the component h:

$$F_h = F_{h-1} - b_h t_{(new)h} q^T_{(new)h}\ ;\ Y = F_0$$

Step 16. Replace X and Y with their corresponding residual matrices E_h and F_h :

$$X = E_h\ ;\ Y = F_h$$

Step 17. Go to Step 1 to implement the procedure for the next component.

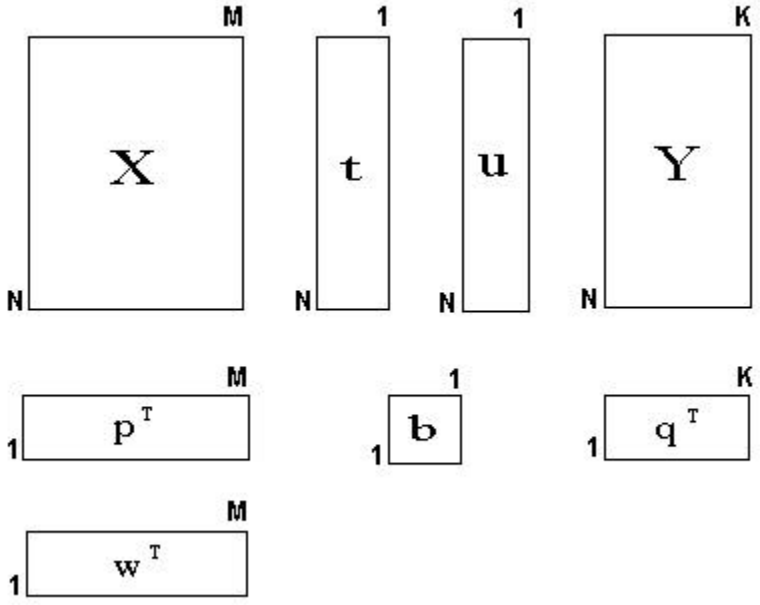


Figure B.1 A Graphical Representation of the Matrices and Vectors used in PLS Algorithm

APPENDIX C

Full Validation Results of GAFEAT-PLS

In this Appendix, full validation approach was applied to GAFEAT-PLS feature selection with 10-fold and Leave-One-Out cross-validation for the Lombardo and HIV_{rt} datasets.

In cross-validation, a dataset is divided into **k** subsets of approximately equal size. A **k**-fold-cross-validation procedure requires the fitting the model into dataset **k** times; at each time a subset is left out once, and only once, as an external test (validation) set and the remaining datasets are combined as a training set. If **k** is equal to the number of data points, it is called Leave-One-Out-Cross-validation.

In the full cross-validation of GAFEAT-PLS, GAFEAT-PLS performed feature selection **k** times (e.g. as many times as the number of cross-validation subsets) for a given dataset and at each time, the reduced training dataset (training dataset with only selected features) was modeled with the Partial Least Squares (PLS) regression. Then, the performance of the model was measured on the corresponding reduced external test set (test set with only selected features) in terms of Q^2 statistics. The PLS models described in this appendix used four latent variables for all calculations. GAFEAT-PLS was executed to select 30 descriptive features from each dataset. The PLS models obtained with feature subsets selected by GAFEAT-PLS were also compared with those of dataset with all features in terms of Q^2 statistics.

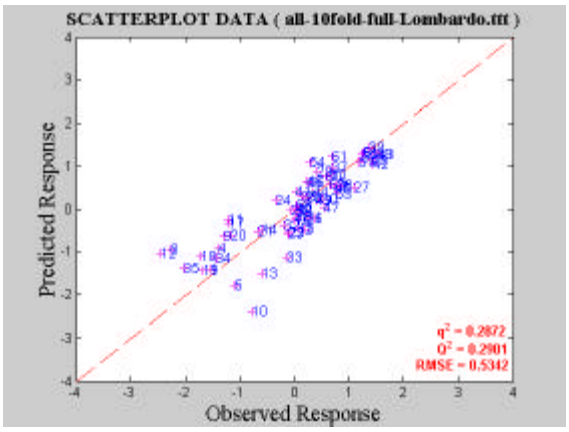
C.1 Results of Full 10-fold Cross-validation of GAFEAT-PLS for the Lombardo Dataset

Table C.1 Parameters of GAFEAT-PLS for the Lombardo Dataset with Full 10-fold Cross-validation. Number of features to be selected is set to 30 features

Population Size	Crossover Probability	Mutation Probability	Maximum # of Generation	Number of Latent Variables
100	0.90	0.02	1000	4
Number of Bootstraps in the Fitness Evaluation		Number of Molecules in the Training Set		Number of Molecules in the Validation Set
20		Depends on fold		4

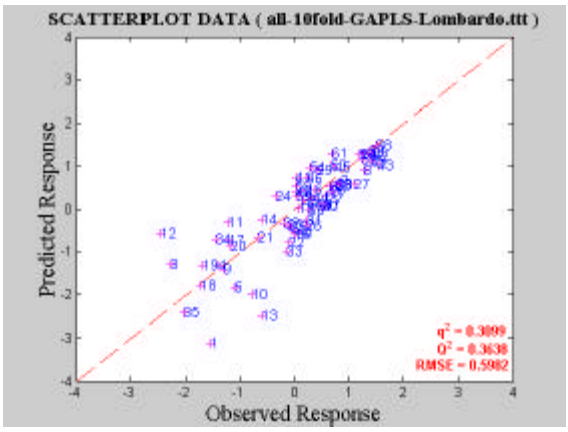
Table C.2 Result of Full 10-fold Cross-validation of GAFEAT-PLS for the Lombardo Dataset

	Full Data (without feature selection)				**GA-PLS** (with feature selection)			
Fold	r^2	R^2	q^2	Q^2	r^2	R^2	q^2	Q^2
1	0.9223	0.9215	0.0999	0.1482	0.9826	0.9824	0.1461	0.1610
2	0.9453	0.9447	0.3106	0.6356	0.9858	0.9853	0.2905	0.5084
3	0.9183	0.9177	0.1239	0.2839	0.9761	0.9761	0.1648	0.1898
4	0.9266	0.9266	0.9435	1.7527	0.9871	0.9870	0.6881	1.1869
5	0.9299	0.9288	0.4619	1.0786	0.9877	0.9867	0.4323	1.1076
6	0.9316	0.9313	0.3161	0.3739	0.9884	0.9884	0.5692	0.8647
7	0.9137	0.9137	0.0644	0.0706	0.9847	0.9846	0.1562	0.6803
8	0.9125	0.9121	0.1214	0.1537	0.9839	0.9837	0.0559	0.0962
9	0.9291	0.9281	0.2072	0.3234	0.9883	0.9881	0.4211	0.4556
10	0.9158	0.9157	0.2951	0.3404	0.9834	0.9832	0.4344	0.5479
Overall			**0.2872**	**0.2901**			**0.3099**	**0.3638**



(A)

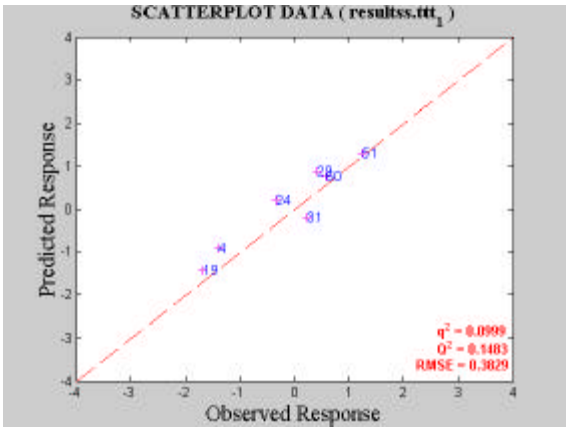
With all features (without feature selection)



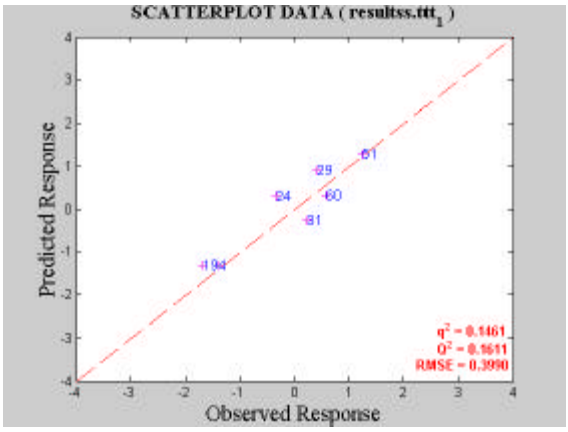
(B)

With GAFEAT-PLS (with feature selection)

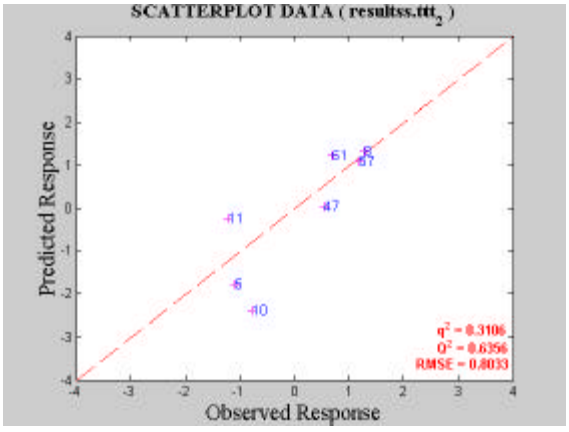
Figure C.1 Overall Results of Full 10-fold Cross-validation for the Lombardo Dataset



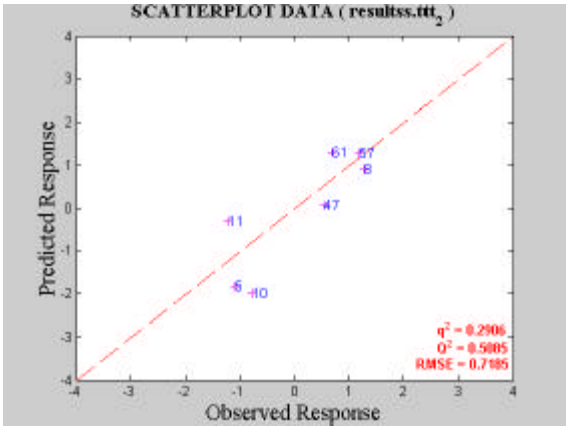
Lombardo Dataset with all features
Result of Fold 1



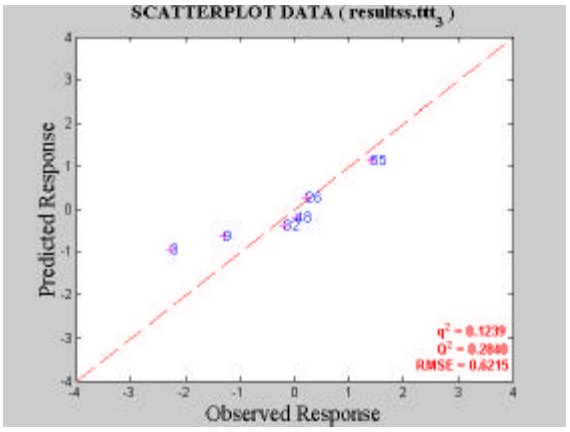
GAFEAT-PLS (Lombardo Dataset)
Result of Fold 1



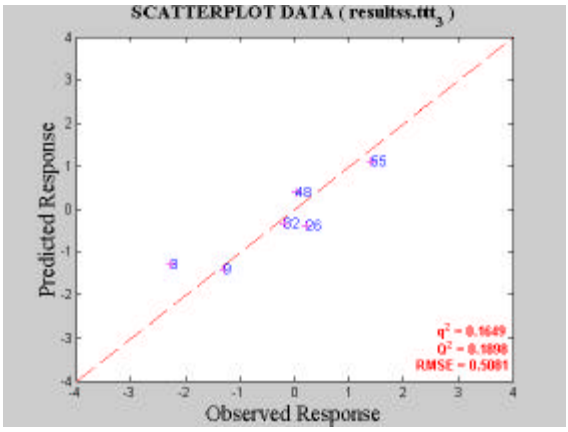
Lombardo Dataset with all features
Result of Fold 2



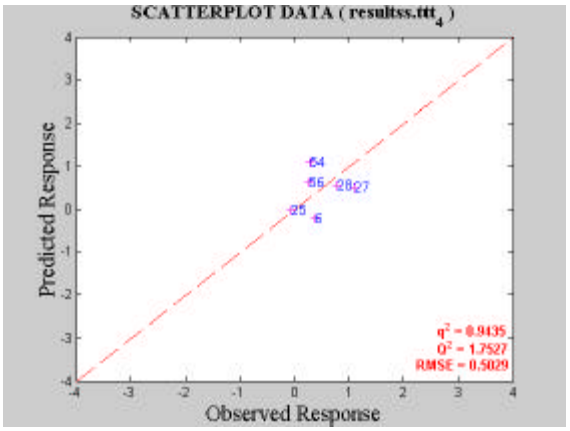
GAFEAT-PLS (Lombardo Dataset)
Result of Fold 2



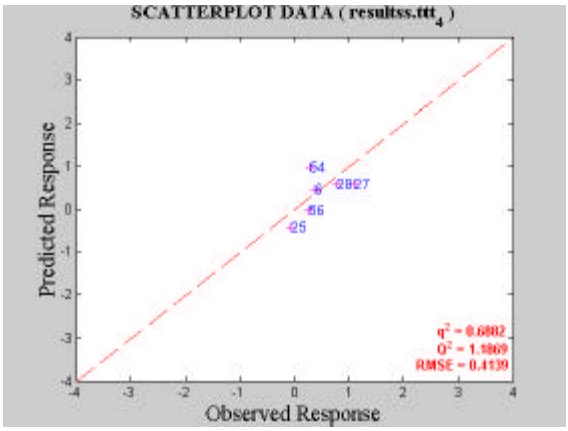
Lombardo Dataset with all features
Result of Fold 3



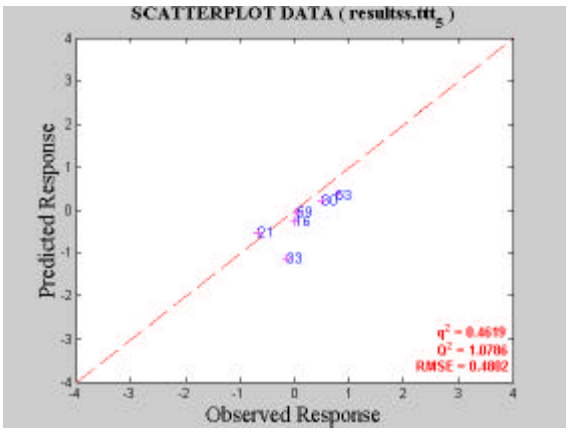
GAFEAT-PLS (Lombardo Dataset)
Result of Fold 3



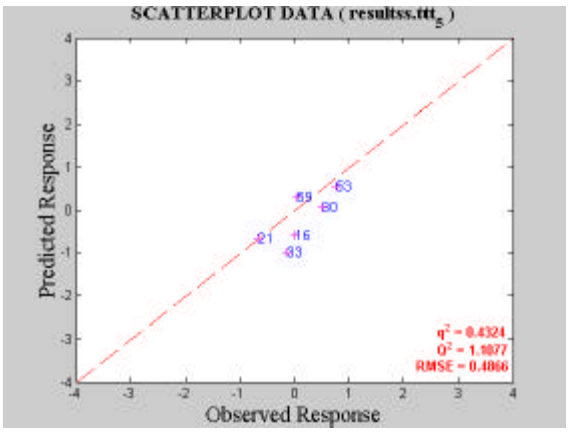
Lombardo Dataset with all features
Result of Fold 4



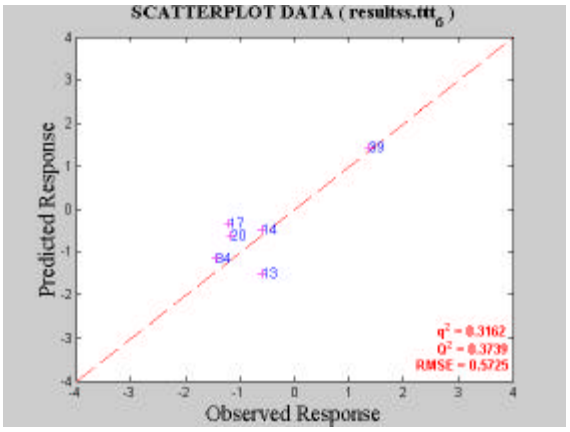
GAFEAT-PLS (Lombardo Dataset)
Result of Fold 4



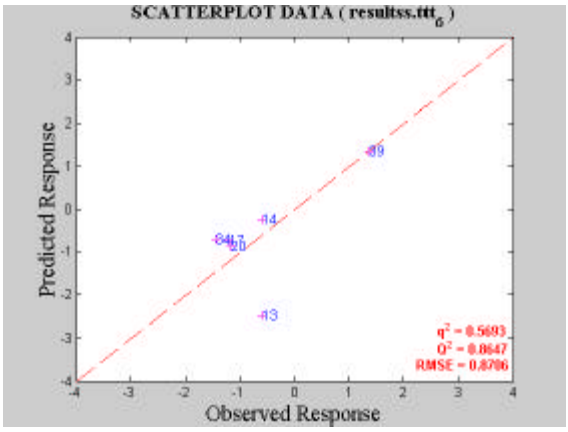
Lombardo Dataset with all features
Result of Fold 5



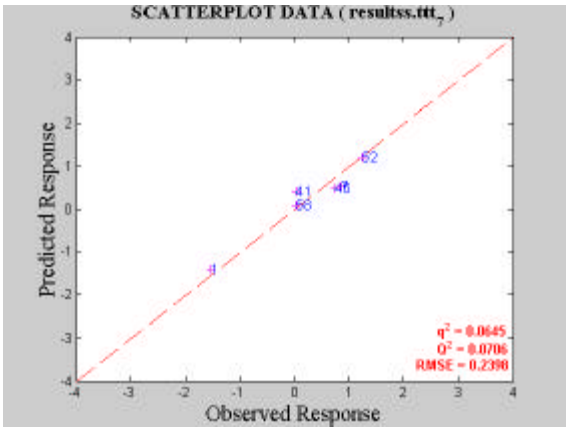
GAFEAT-PLS (Lombardo Dataset)
Result of Fold 5



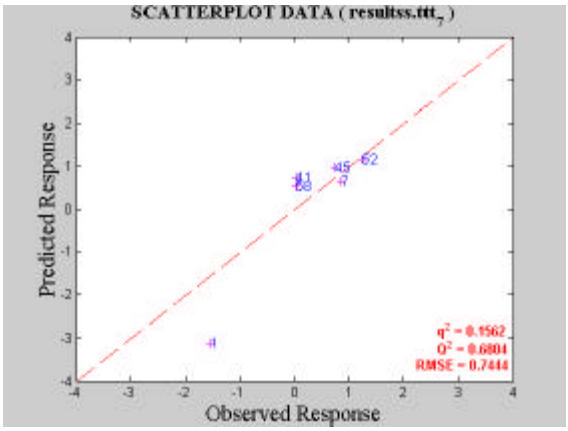
Lombardo Dataset with all features
Result of Fold 6



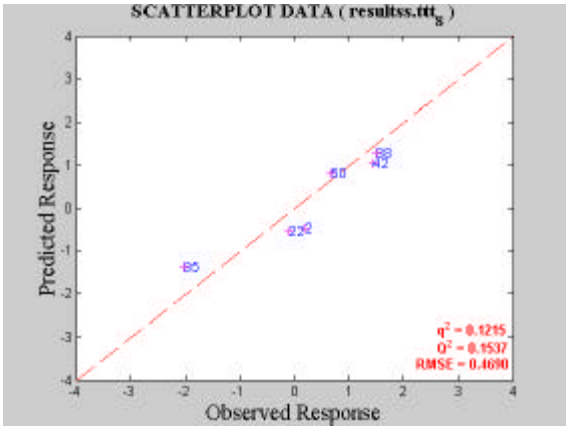
GAFEAT-PLS (Lombardo Dataset)
Result of Fold 6



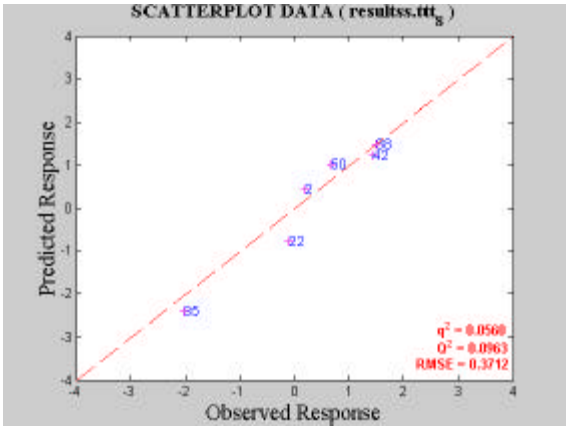
Lombardo Dataset with all features
Result of Fold 7



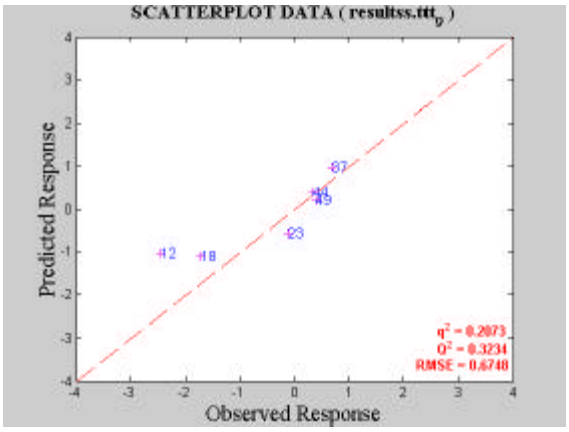
GAFEAT-PLS (Lombardo Dataset)
Result of Fold 7



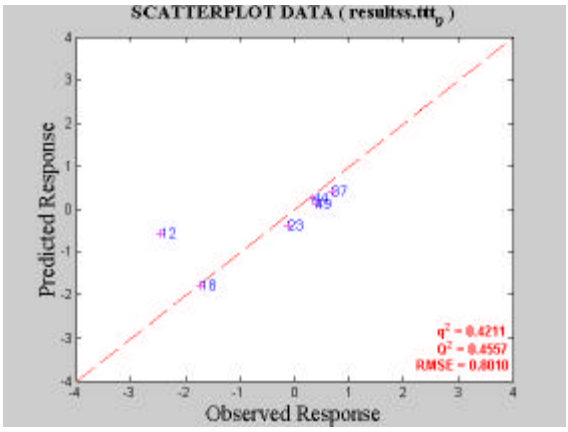
Lombardo Dataset with all features
Result of Fold 8



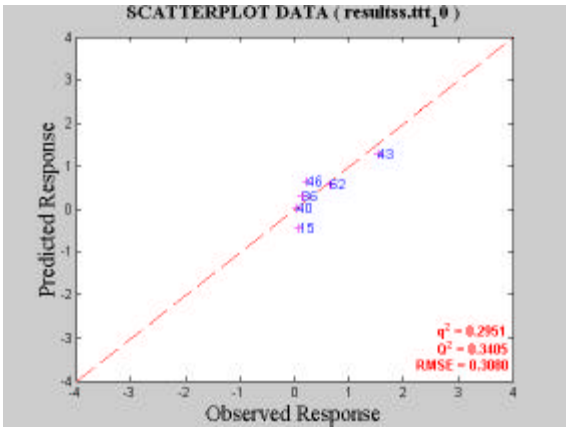
GAFEAT-PLS (Lombardo Dataset)
Result of Fold 8



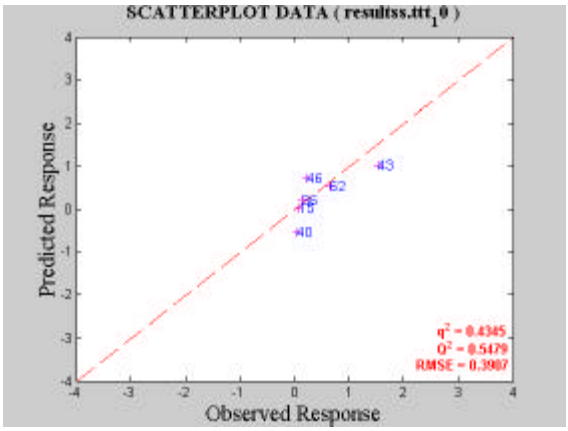
Lombardo Dataset with all features
Result of Fold 9



GAFEAT-PLS (Lombardo Dataset)
Result of Fold 9



Lombardo Dataset with all features
Result of Fold 10



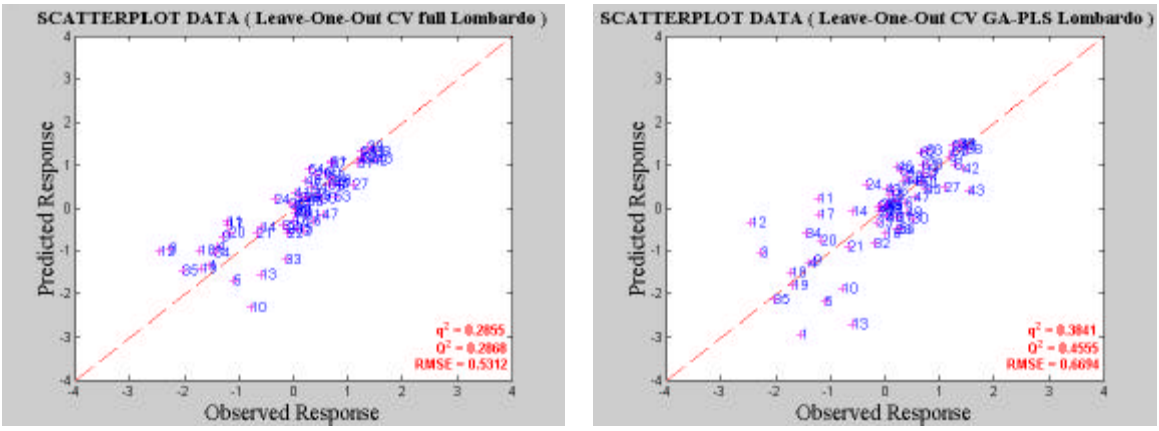
GAFEAT-PLS (Lombardo Dataset)
Result of Fold 10

Figure C.2 Comparison of the Individual Fold of the Full 10-fold Cross-validation of GAFEAT-PLS with Full Dataset

C.2 Results of Full Leave-One-Out Cross-validation of GAFEAT-PLS for the Lombardo Dataset

Table C.3 Parameters of GAFEAT-PLS for the Lombardo Dataset with Full Leave-One-Out Cross-validation. Number of features to be selected is set to 30 features.

Population Size	Crossover Probability	Mutation Probability	Maximum # of Generation	Number of Latent Variables
100	0.90	0.02	1000	4
Number of Bootstraps in the Fitness Evaluation		Number of Molecules in the Training Set		Number of Molecules in the Validation Set
20		56		6



(A) With all features (without feature selection) **(B)** With GAFEAT-PLS (with feature selection)

Figure C.3 Overall results of Leave-One-Out Cross-validation for the Lombardo Dataset

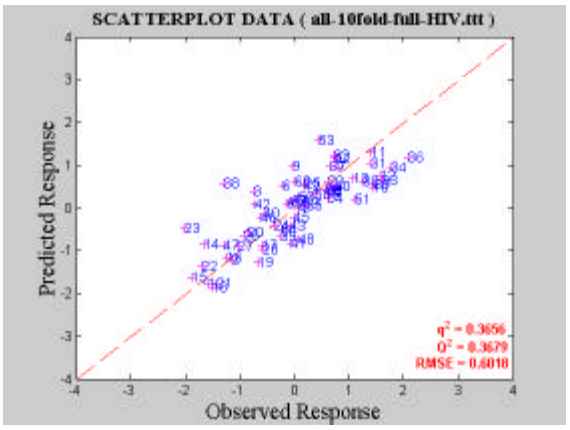
C.3 Results of Full 10-fold Cross-validation of GAFEAT-PLS for the HIV_{rt} Dataset

Table C.4 Parameters of GAFEAT-PLS for the HIV_{rt} Dataset with Full 10-fold Cross-validation. Number of features to be selected is set to 30 features.

Population Size	Crossover Probability	Mutation Probability	Maximum # of Generation	Number of Latent Variables
100	0.90	0.02	1000	4
Number of Bootstraps in the Fitness Evaluation		Number of Molecules in the Training Set		Number of Molecules in the Validation Set
20		Depends on fold		4

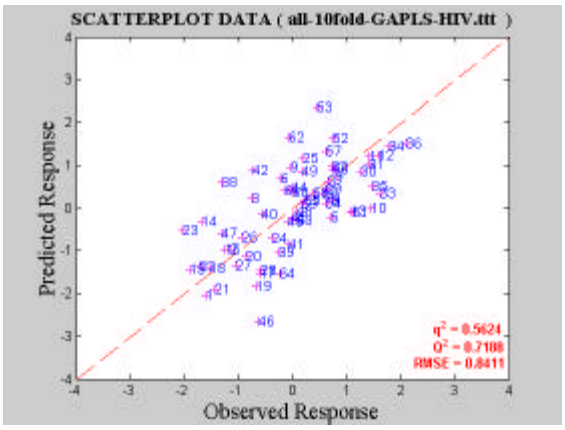
Table C.5 Result of Full 10-fold Validation of GAFEAT-PLS for the HIV_{rt} Dataset

	Full Data (without feature selection)				**GA-PLS** (with feature selection)			
Fold	$\mathbf{r^2}$	$\mathbf{R^2}$	$\mathbf{q^2}$	$\mathbf{Q^2}$	$\mathbf{r^2}$	$\mathbf{R^2}$	$\mathbf{q^2}$	$\mathbf{Q^2}$
1	0.8146	0.8131	0.2269	0.3996	0.9521	0.9508	0.4839	0.6375
2	0.8348	0.8336	0.7033	1.2646	0.9607	0.9583	0.7483	1.5513
3	0.7917	0.7915	0.1504	0.2119	0.9502	0.9501	0.3207	0.3232
4	0.8164	0.8161	0.4342	0.6128	0.9475	0.9473	0.3933	0.7025
5	0.7991	0.7991	0.3047	0.3243	0.9474	0.9474	0.7488	1.6040
6	0.7876	0.7849	0.4844	1.1770	0.9428	0.9408	0.2196	0.7333
7	0.8289	0.8289	0.7157	0.7335	0.9616	0.9609	0.8538	1.0954
8	0.8221	0.8209	0.1055	0.5683	0.9508	0.9481	0.2640	1.4951
9	0.8233	0.8222	0.2128	0.6086	0.9401	0.9399	0.6387	3.7095
10	0.8172	0.8164	0.1687	0.3187	0.9508	0.9498	0.6328	0.7168
Overall			**0.3656**	**0.3679**			**0.5624**	**0.7188**



(A)

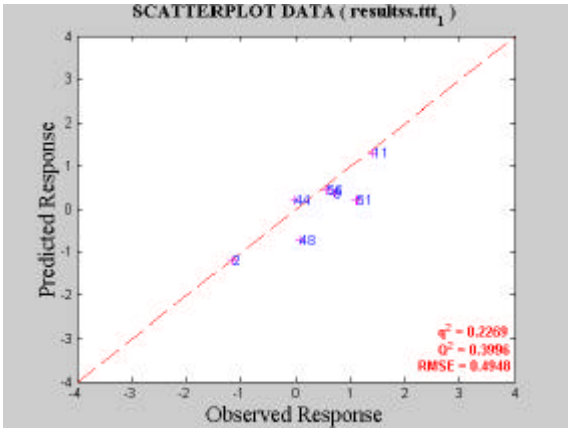
With all features (without feature selection)



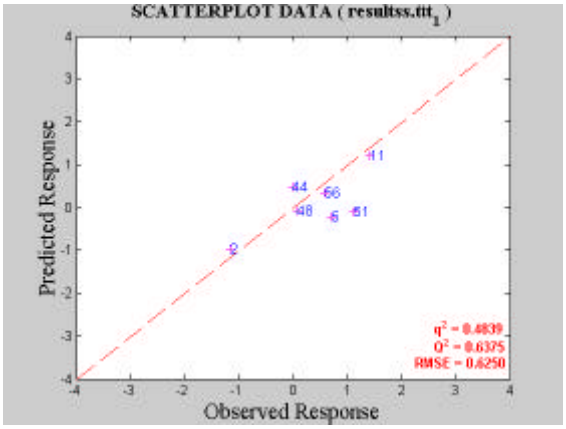
(B)

With GAFEAT-PLS (with feature selection)

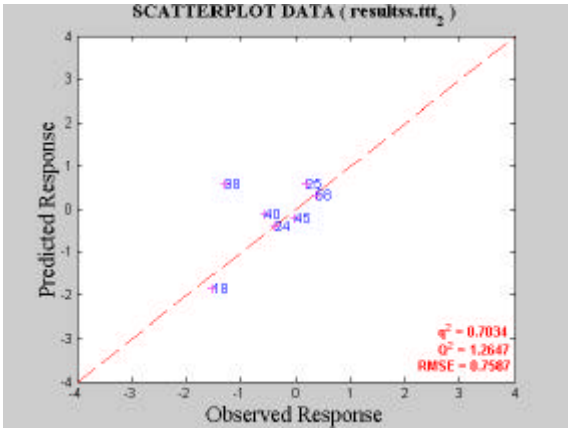
Figure C.4 Overall Results of Full 10-fold Cross-validation for the HIV_{rt} Dataset



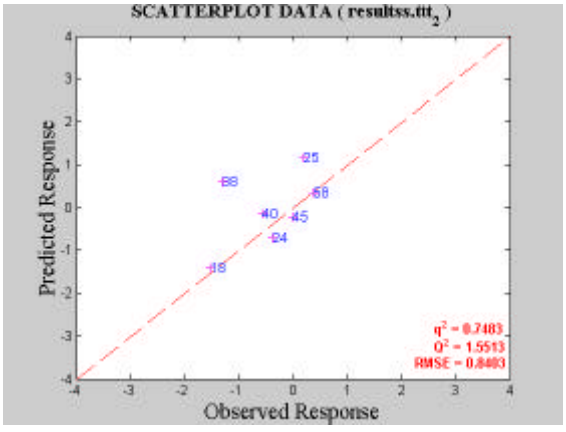
HIV_{rt} Dataset with all features
Result of Fold 1



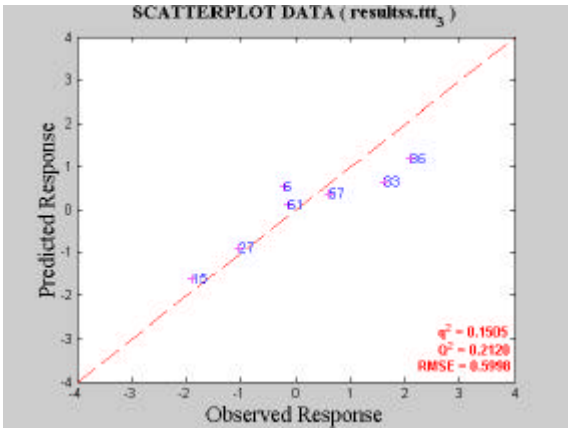
GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 1



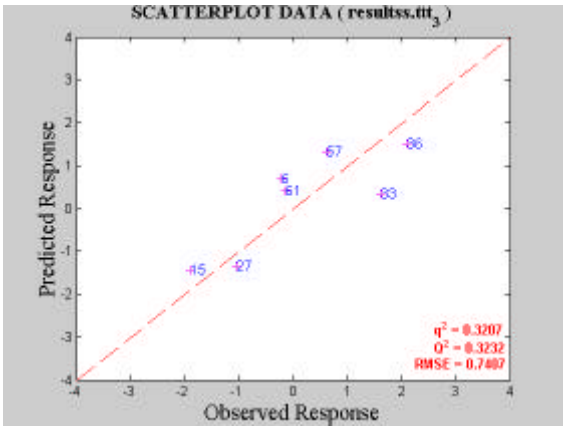
HIV_{rt} Dataset with all features
Result of Fold 2



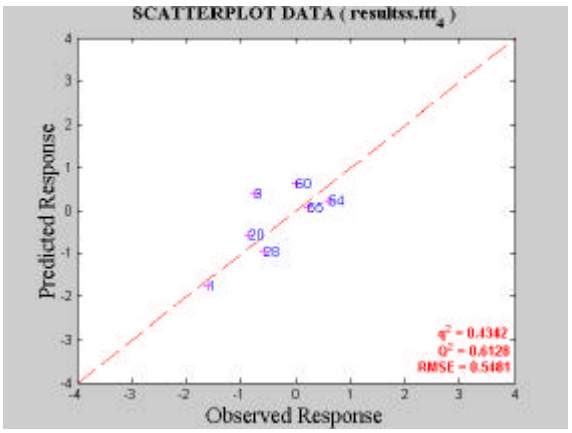
GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 2



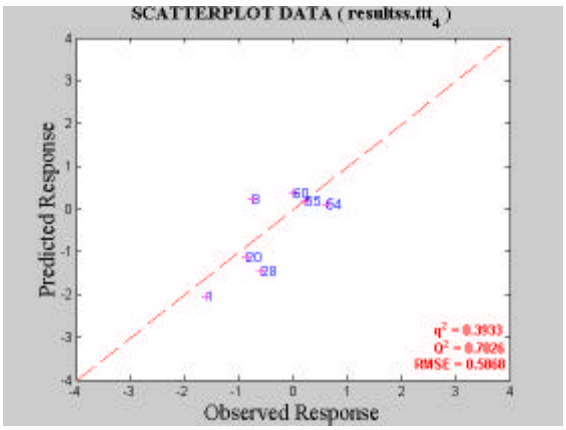
HIV_{rt} Dataset with all features
Result of Fold 3



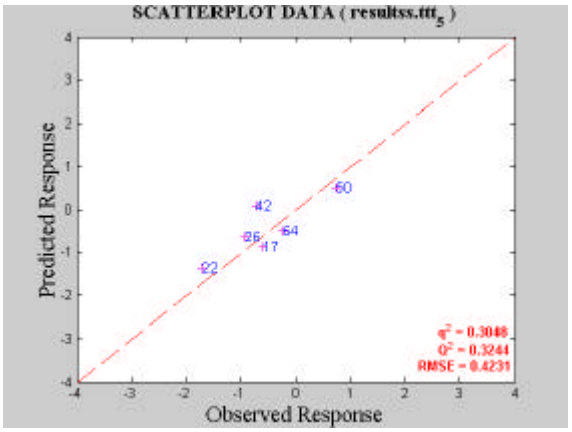
GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 3



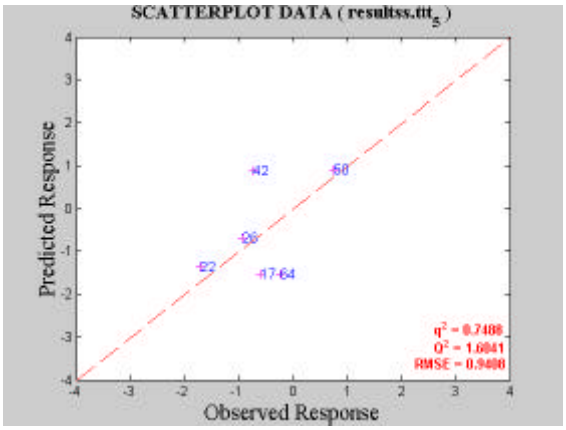
HIV_{rt} Dataset with all features
Result of Fold 4



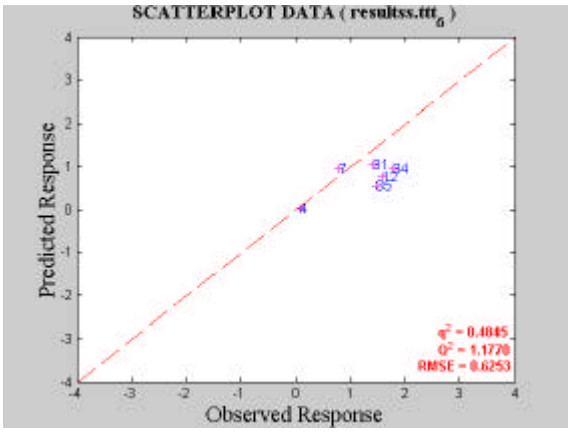
GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 4



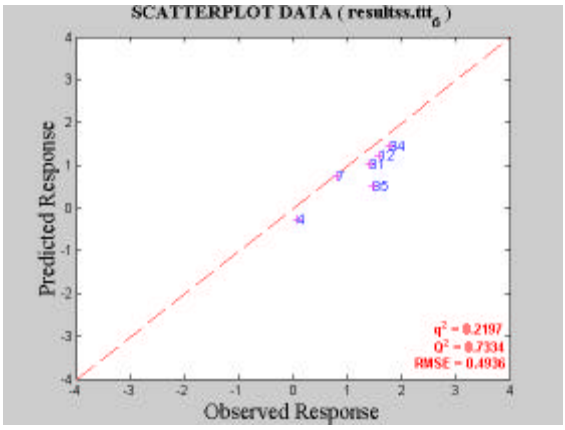
HIV_{rt} Dataset with all features
Result of Fold 5



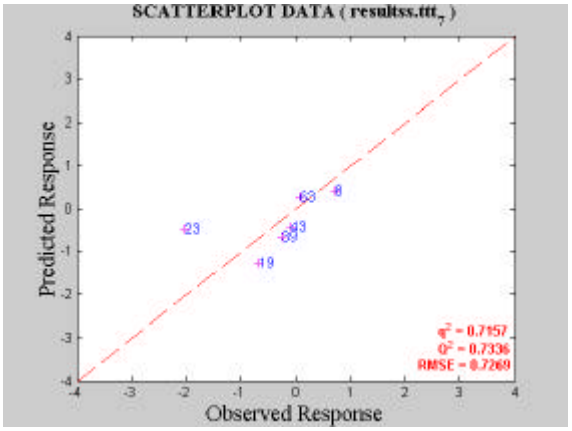
GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 5



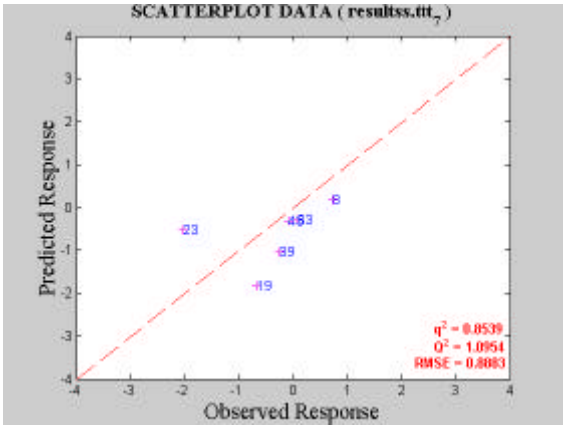
HIV_{rt} Dataset with all features
Result of Fold 6



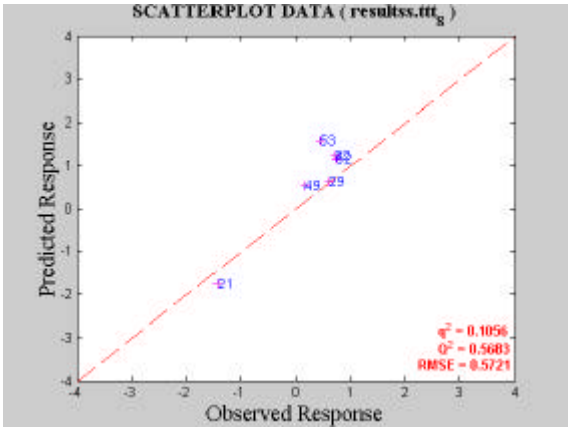
GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 6



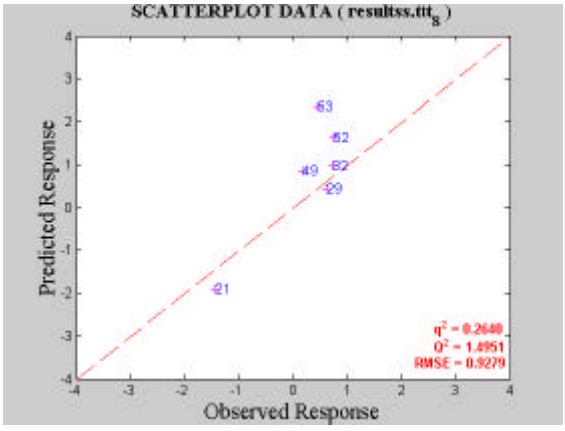
HIV_{rt} Dataset with all features
Result of Fold 7



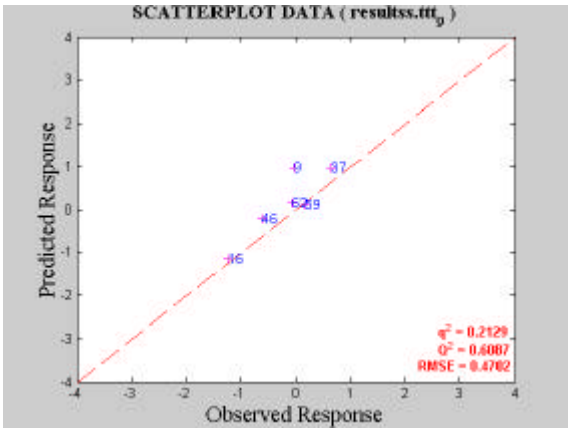
GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 7



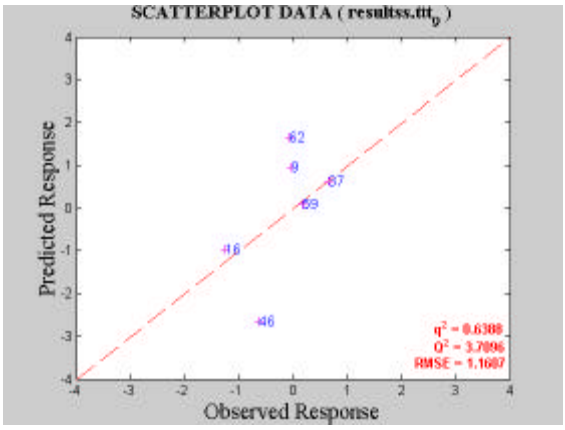
HIV_{rt} Dataset with all features
Result of Fold 8



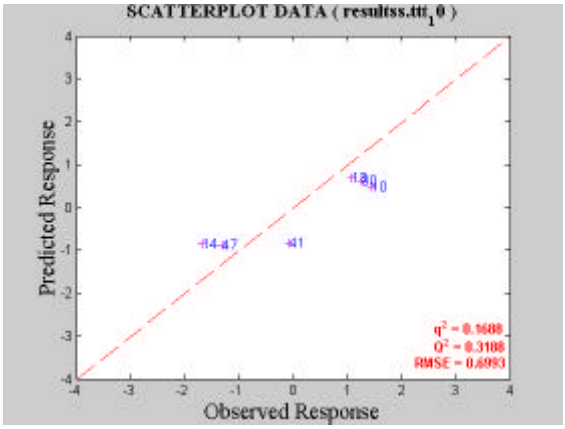
GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 8



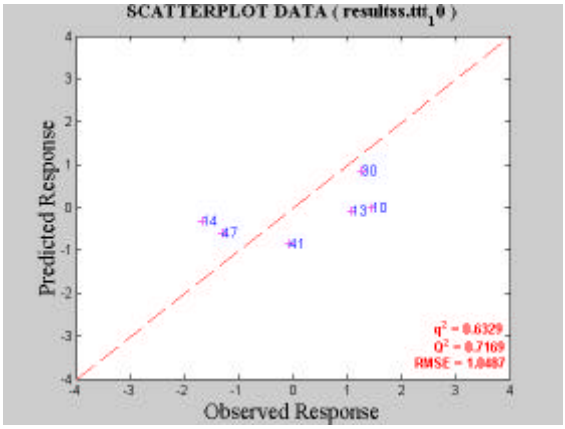
HIV_{rt} Dataset with all features
Result of Fold 9



GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 9



HIV_{rt} Dataset with all features
Result of Fold 10



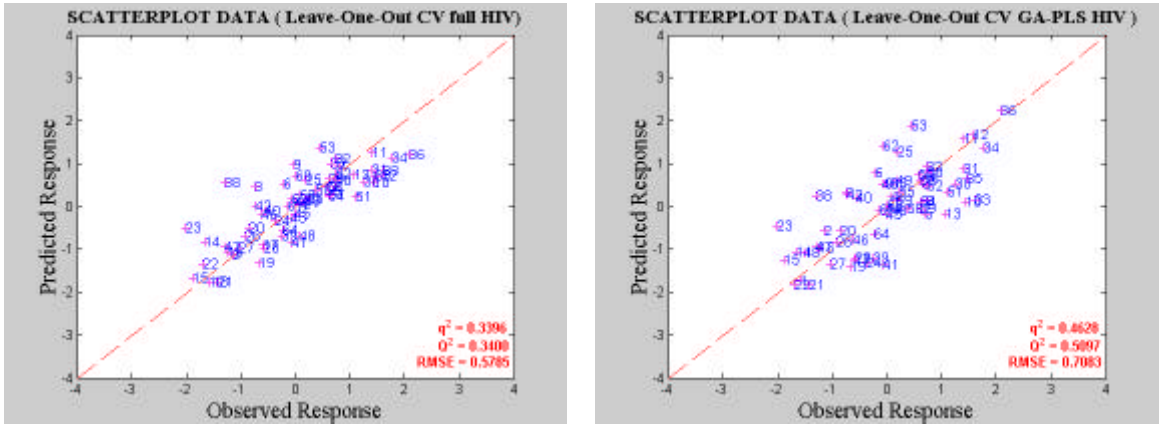
GAFEAT-PLS (HIV_{rt} Dataset)
Result of Fold 10

Figure C.5 Comparison of the Individual Fold of the Full 10-fold Cross-validation of GAFEAT-PLS with Full Dataset

C.4 Results of Full Leave-One-Out Cross-validation of GAFEAT-PLS for the HIV_{rt} Dataset

Table C.6 Parameters of GAFEAT-PLS for the HIV_{rt} Dataset with Full Leave-One-Out Cross-validation. Number of features to be selected is set to 30 features.

Population Size	Crossover Probability	Mutation Probability	Maximum # of Generation	Number of Latent Variables
100	0.90	0.02	1000	4
Number of Bootstraps in the Fitness Evaluation		Number of Molecules in the Training Set		Number of Molecules in the Validation Set
20		58		6



(A) All features (without feature selection)

(B) GAFEAT-PLS (with feature selection)

Figure C.6 Overall results of Leave-One-Out Cross-validation for the HIV_{rt} Dataset

APPENDIX D

Comparison of the Proposed Evolutionary Algorithms for Feature Selection with CART

In this Appendix, the feature subsets selected from the Lombardo and HIV_{rt} datasets by the proposed evolutionary algorithms (GAFEAT-PLS, GAFEAT-PLS with INLR, and GAFEAT-LL) will be compared with that of CART (**C**lassification **A**nd **R**egression **T**rees) method. The following brief information about CART was taken from [214].

D.1 CART Methodology

CART is a statistical procedure introduced by Breiman, Friedman, Olshen, and C. Stone in 1984 [215]. CART procedure can be used to analyze either categorical (classification problem) or continuous data (regression problem). The CART methodology is known as binary recursive partitioning. The process is binary since parent nodes are always split into exactly two child nodes. The process is recursive since it can be repeated by treating each child node as a parent. The key elements of a CART analysis are a set of rules for:

i) Splitting each node in a tree;

ii) Deciding when a tree is complete; and

iii) Assigning each terminal node to a class outcome for classification problems (or predicted value for regression problem).

CART presents its results in the form of decision trees and therefore it is a significant departure from more traditional statistical analysis procedure. CART uses a

decision tree to display how data may be classified or predicted. By asking a series of "yes/no" questions concerning database fields, CART automatically searches for important relationships and uncovers hidden structure even in highly complex datasets.

D.2 Feature Selection with CART

CART is often employed to select a manageable number of relevant features from datasets with hundreds of features. Since one of the goals of CART is to develop a simple tree structure for data, relatively few variables may appear explicitly in the splitting criteria. This can be interpreted that other variables are not important in understanding or predicting the response variable. Unlike a linear regression model, a variable in CART can be considered highly important even if it never appears as a primary node splitter, since CART keeps track of surrogate splits in the tree-growing process and the contribution a variable can make in prediction is not determined only by primary splits. The importance score of a variable in CART is calculated in the following way:

CART looks at the improvement measure attributable to each variable in its role as a surrogate to the primary split. The value of these improvements are summed over each node and totaled, and scaled relative to the best performing variable. The variable with the highest sum of improvement is scored to 100, and all other variables will have lower scores ranging downwards towards to zero.

It is noted that the importance score measures a variable's ability to mimic the chosen tree and to play a role as a stand-in for variables appearing in primary splits. The importance score does not say anything about the value of any variable in the construction of other trees. Therefore, the importance score does not indicate an absolute

information value of a variable and ranking of the variables are strictly relative to a given tree structure.

D.3 CART Analysis for the Lombardo and HIV_{rt} Datasets

CART program was used to select important features from the Lombardo and HIV_{rt} datasets. The default setting of the CART program was used to analyze these QSAR datasets. The importance of the variables (features) was obtained from the best CART tree for both datasets. 11 variables for the Lombardo and 12 variables for the HIV_{rt} were identified as important features based on the best CART tree. Important variables for the Lombardo and HIV_{rt} datasets were presented in the Tables D.1 and D.2, respectively.

Table D.1 Importance of Variables of the Lombardo Dataset Calculated from the Best CART Tree

Variable ID	Variable Name	Score	
V16	AbsDRN7	100.00	
V2	drns1	92.77	
V1	SHWHBD_358	82.04	
V24	dknd3	74.91	
V4	AbsDRN5	72.25	
V43	AbsL7	68.37	
V199	lapld7	16.46	
V167	AbsDRN1	15.96	
V95	lapld8	15.96	
V28	Gmax_100	13.49	
V294	fuks3	11.87	

Table D.2 Importance of Variables of the HIV_{rt} Dataset Calculated from the Best CART Tree

Variable ID	Variable Name	Score	
V17	pips8	100.00	
V88	pips1	90.83	
V1	Del.G.NIA	78.95	
V3	pips6	62.49	
V28	HMax.99	51.69	
V8	SHsOH.91	51.69	
V172	AbsBNP7	44.43	
V18	SaasC.119	30.15	
V38	AbsDRN4	28.15	
V15	SssCH2.111	25.46	
V32	dxp4.49	25.21	
V149	CssCH2.202	23.94	

The performances of the feature subsets selected by CART on PLS and SVM regression models were calculated based on 100-bootstrap sample by leaving 6 and 8 molecules in the validation sets for the Lombardo and HIV_{rt} datasets, respectively. These results are present in Tables D.3 and D.4 for the Lombardo and HIV_{rt} datasets, respectively.

Table D.3 100-bootstrap Validation for the Selected 11 Features from the Lombardo Dataset by CART

Learning Models	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
PLS	0.7723	0.7723	0.3112	0.3129
SVM	0.8721	0.8712	0.2332	0.2333

Table D.4 100-bootstrap Validation for the Selected 12 Features from the HIV_{rt} Dataset by CART

Learning Models	Training Error		Validation Error	
	r^2	R^2	q^2	Q^2
PLS	0.6535	0.6534	0.5240	0.5415
SVM	0.8987	0.8953	0.2987	0.3058

D.4 Comparison of CART with the Proposed Feature Selection Methods

In order to compare the feature subsets selected by CART with those of the proposed evolutionary algorithms (GAFEAT-PLS, GAFEAT-PLS with INLR, and GAFEAT-LL), SVM regression was used to model the CART feature subsets. Since CART selected 11 features from the Lombardo and 12 features from HIV_{rt} datasets, the proposed feature selection algorithms were executed to select feature subsets of size 10 for both datasets for a fair comparison. Tables D.5 and D.6 present the comparison of feature selection methods based on SVM regression model with 100-bootstrap validation

for the Lombardo and HIV_{rt} datasets, respectively. Figures D.1 and D.2 also depict these results graphically.

Table D.5 Comparison of Feature Selection Methods based on SVM regression Model with 100-bootstrap Validation for the Lombardo Dataset

Feature Selection Method	Number of Selected Features	Q^2_{SVM}
GAFEAT-LL	**10**	0.3016
GAFEAT-PLS	**10**	0.1951
GAFEAT-PLS with INLR	**10**	0.1951
CART	**11**	0.2333

Table D.6 Comparison of Feature Selection Methods based on SVM regression Model with 100-bootstrap validation for the HIV_{rt} Dataset

Feature Selection Method	Number of Selected Features	Q^2_{SVM}
GAFEAT-LL	**10**	0.2237
GAFEAT-PLS	**10**	0.2218
GAFEAT-PLS with INLR	**10**	0.1957
CART	**12**	0.3058

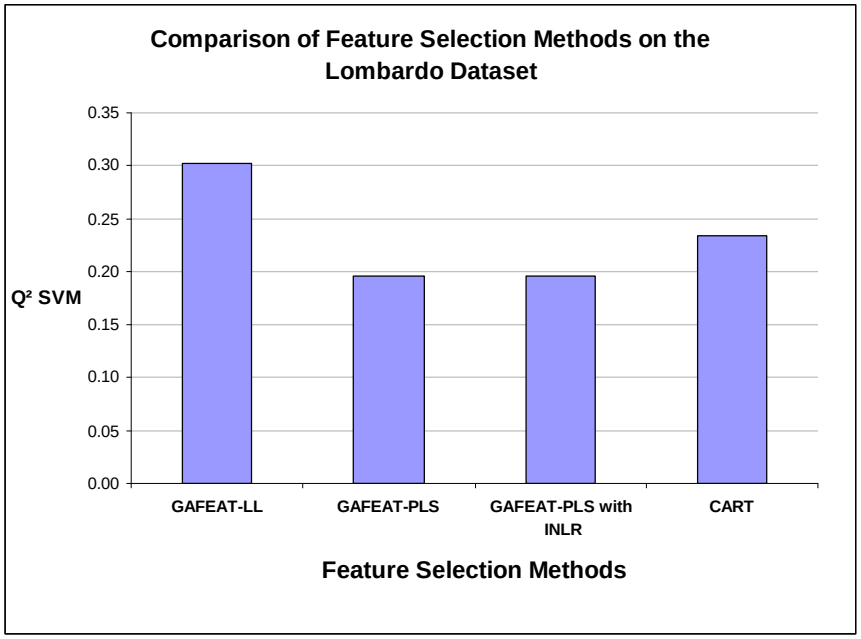


Figure D.1 Comparison of Feature Selection Methods based on SVM regression Model with 100-bootstrap Validation for the Lombardo Dataset

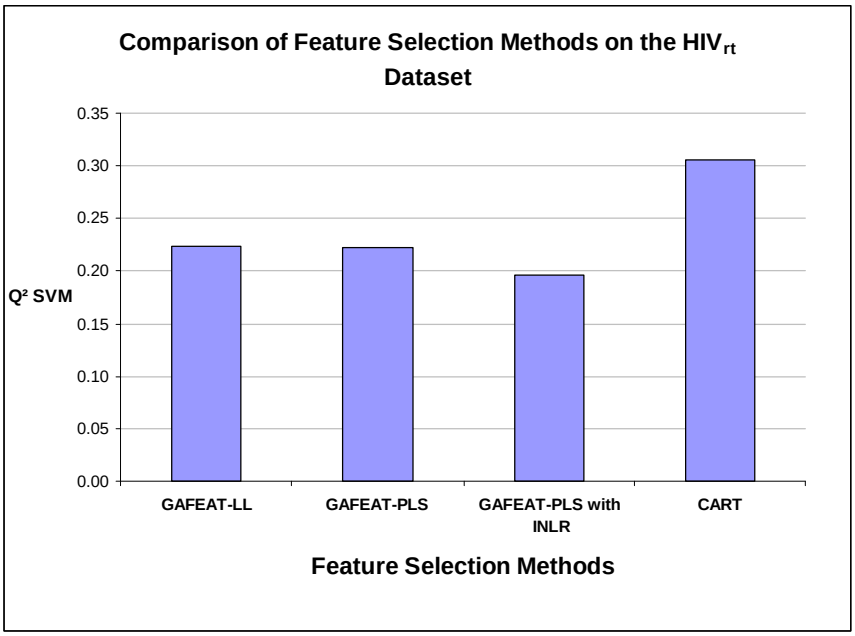


Figure D.2 Comparison of Feature Selection Methods based on SVM regression Model with 100-bootstrap Validation for the HIV_{rt} Dataset

It is clear from figures D.1 and D.2 that the proposed feature selection methods give comparable, or even better, results than those of CART. It should be noted that even if the subsets of size 10 selected by the proposed feature selection algorithms are not the best subset for the Lombardo and HIV_{rt} datasets, these subsets still give comparable results to that of CART.