This is to certify that the
dissertation entitled

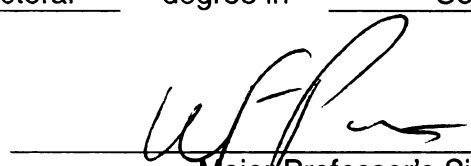
Evolutionary Optimization and Ensemble Techniques for
Data Mining and Pattern Recognition

presented by

Alexander P. Topchy

has been accepted towards fulfillment
of the requirements for the

Doctoral degree in Computer Science



Major Professor's Signature

5/12/04

Date

EVOLUTIONARY OPTIMIZATION AND ENSEMBLE TECHNIQUES FOR DATA MINING AND PATTERN RECOGNITION

By

Alexander P. Topchy

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2004

ABSTRACT

EVOLUTIONARY OPTIMIZATION AND ENSEMBLE TECHNIQUES FOR DATA MINING AND PATTERN RECOGNITION

By

Alexander P. Topchy

This dissertation addresses fundamental data mining and pattern recognition problems – feature extraction, modeling, and data clustering – through evolutionary computation and ensemble-based approaches.

We offer feature extraction methods for improved pattern classification using genetic algorithms. New features are synthesized by merging the values of original variables during the search process. The genetic search of (sub-) optimal combinations of values is performed using a graph-based encoding of candidate solutions. A compact solution representation with minimal redundancy is used for a wide class of grouping problems, including clustering of variable values. Genetic value clustering is applied to text categorization, DNA-based assignments of individuals in population genetics and parametric learning of Bayesian network classifiers. It is shown that such feature extraction results in better predictive accuracy of classification decisions.

We develop genetic programming algorithms for modeling input-output mappings of continuous variables that incorporates dynamical fitting of free parameters of evolved models. Traditional genetic programming is extended by gradient descent optimization of leaf coefficients of tree-like programs during the evolutionary search that is made

possible using algorithmic differentiation. Experimental results show significant improvement in both computational requirements and modeling accuracy for a set of symbolic regression problems.

Ensembles of partitions of data sets are studied in two respects: combination of multiple clusterings and generation of clusterings for an ensemble. We develop two efficient consensus functions for finding a combined partition of good quality. The first consensus function uses an information-theoretic principle based on maximal generalized mutual information. The second function finds a consensus clustering by estimating a probabilistic mixture model from the observed ensemble. It is demonstrated that the ensemble's partitions can be generated by weak clustering algorithms, in particular, by clustering in random low-dimensional subspaces of the original feature space. Experiments indicate that ensemble of an weak partitions can be more accurate than a single sophisticated clustering algorithm. Finally, we consider how the partition generation process can be made adaptable to provide better decisions for the patterns located near the inter-cluster boundaries.

Dedicated to my parents, Pavel Topchy and Tamara Zemel

Acknowledgements

The years at MSU have been a fantastic experience and it is a joy to remember and to thank my teachers and colleagues whose influence contributed to this thesis.

First and foremost, I would like to thank my advisor, Bill Punch, for his extraordinary support and patience. His fruitful suggestions and our discussions contributed enormously to this work. From the first time I stepped in the door of Computer Science Department, until now, Bill has allowed me freedom to explore my own directions in research. I have come to appreciate the wisdom of his way that has guided me safely through the process.

I owe many inspirational ideas to Anil Jain, who has been like a second advisor to me. It was a privilege to be a student in his pattern recognition classes, which defined my interest to the field for many years to come. The results of this thesis would be impossible without Anil Jain's motivation and without research assistantship he kindly provided in the past year.

Kim Scribner fostered my interest in genetic data analysis. I deeply appreciate his enthusiasm for unorthodox approaches, the high standards he set, the genuinely positive attitude to science and to my research progress. He supported me materially and morally for many years for what I am very grateful.

I thank Erik Goodman for asking just the right questions to improve the work, for providing valuable feedback on my projects at all times, for unforgettable GARAGe group meetings. Most importantly, thanks to his help with organizing and hosting my visit to MSU in the summer of 1997, I became determined to pursue the study at MSU, in particular, in GARAGE group, which eventually became my alma mater.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Contemporary data mining research stems from the three fundamental information-processing tasks: classification, clustering and prediction. These problems are closely related in their mathematical formulations as well as their methods for solving those tasks. Knowledge domains shape unique characteristics of each particular problem, but underlying principles of solution and decision-making tools often remain the same. Mathematical optimization is one of the ubiquitous methods used for such information processing. Numerical and combinatorial optimization is essential for achievements in applied data mining, such as genetic data analysis, e-commerce knowledge extraction, or autonomous robot navigation. One such contribution to data mining has been made by evolutionary computations, as reviewed recently in [42]. Evolutionary search has grown to become a discipline in its own right with major interactions from system identification, statistical pattern recognition, design, pure discrete mathematics, and many other fields.

New ensemble methods have recently become important not only in traditional statistical pattern recognition [28], but in the emerging area of distributed data mining

[101][67]. Multiple classifiers (ensembles of), clusterings and regression solutions can be combined to contribute to an improved overall consensus solution.

Both evolutionary and ensemble methods operate with multiple solutions for a given problem, though in different ways. In evolutionary computation, we rely on information exchange between candidate solutions to find a (sub-) optimal solution as a result of a stochastic dynamic process. In ensembles of classifiers/clusterings, the ultimate solution is produced by a fixed known consensus function. Yet all such methods implicitly or explicitly seek to combine strengths and benefits of individual solutions.

1.1 Motivation, goals, problem domains

This dissertation addresses three problems, which are central to modern data mining:

1. Finding the best subset of variables relevant to a computational model. In general, new variables can be synthesized from existing data features. Generalization or predictive classification accuracy of the model is the main evaluation criterion in feature selection and extraction.
2. Data model identification. Prediction of output variables is based on learning of an unknown real-valued function for a given training sample and a set of elementary functional components.
3. Data clustering. Partitioning of data into meaningful groups of data points. Clustering criteria use available data features and a variety of prior knowledge about data distribution properties.

Generally, these problems are known to be NP-hard. Therefore, solutions are sou using heuristics in combination with powerful search methods. Despite signifi progress, every domain problem has its own set of difficulties that are not satisfactorily resolved. This dissertation work contributes to data mining solut generally and more narrowly concentrates on the following central issues of the ab mentioned tasks:

1. In classification: evolutionary feature extraction and dimensionality reduc based on genetic search of optimal feature value combinations. Accuracy-dri construction of new features using intelligent value clustering.
2. In model identification: fast genetic programming for data-driven induction predictive models with real-valued input-output mappings.
3. In clustering: generic and robust methods to combine multiple clusterings of including the use of novel fast algorithms for consensus clustering as wel inexpensive and weak clustering components.

The following introduces these tasks and highlights the role of evolutionary and ensem computations in their solutions.

The data models that we consider are used for classification and predic systems. The fidelity of the model can be judged by its accuracy or other related meas Our ultimate goal is to improve the predictive accuracy of these computational model

The abundance of features is characteristic of practical tasks such as ir recognition or text classification. The measurement complexity and associated costs f researchers to seek a balance between representing all the data and representing only data necessary to solve the problem. Other constraints exist as well, such as the fact

the extracted data must be comprehensible or compact. Statistical pattern recognition a machine learning often define these problems as *feature subset selection* and *featu extraction*, where features refer to the components of the pattern measurement vect The process of feature selection/extraction is also known as dimensionality reducti since the dimensionality of the pattern vector is reduced. The result of dimensional reduction could be much richer than simplified data and knowledge representation – can also help diminish the effect of the "curse of dimensionality" [28][133]. The curse dimensionality phenomenon manifests itself in decreasing classification accuracy as t number of features grows beyond a certain limit. Such decreasing accuracy occurs wh parameters of the model are estimated from the data. Estimation errors will inevitab degrade the accuracy of the model derived from fixed training samples. It is important note that both the number of features and the number of values assumed by ea individual feature are problematic. Thus this total number of values in the input spa called measurement complexity, is the crux issue. Overcoming the curse dimensionality by lowering the measurement complexity is pivotal to this study. general, appropriate input transformation may include the elimination of some featu altogether or only some of the values or both.

1.2 Contributions of the thesis

As the *first main contribution* of the dissertation, we propose a family of algorithms merging both values and features. As such, this work is a kind of feature extracti While numerous approaches for dimensionality reduction are known, we focus on

called wrapper-style algorithms. Wrapper-style feature extraction evaluates candidate solutions while simultaneously doing both model induction and accuracy estimation. It is more computationally expensive than filter-style extraction or data pre-processing but potentially more accurate, as we detail below. The novelty of the approach is in the use of a genetic algorithm for clustering the values of the variables. In contrast, traditional genetic algorithm approaches to data mining operate only on the features as a whole by including/excluding them from the feature set. We take that work a step further and organize search in the entire input space. The role of various search criteria is analyzed and compared against filter-type feature extraction, which does not use an explicit data model, but instead relies only on information-theoretic methods, such as maximization of mutual information between the input and target variables.

Special attention is paid to the design of the genetic algorithm for value clustering. Since variables assume nominal values, the model search is effectively reduced to a grouping problem. This problem is known to be NP-hard, and genetic algorithms often outperform other grouping heuristics developed for bin packing, graph coloring, load balancing, etc. [32]. New genetic representations are designed for the studied family of models.

We analyze the performance of the value clustering algorithms in three application domains:

1. Text categorization. Classification of text using a naïve Bayes classifier and a "bag of words" data model.
2. Bayesian network induction. Improving the accuracy of a Bayesian network classifier by learning the conditional probability tables with local structure.

3. Population genetics. Genotype-based assignment of individuals to populations of origin. The data model is an extension of a multinomial model with multiple features and certain value constraints within the features. The allelic values of multilocus individuals are subjected to merging.

The *second main problem* studied in the thesis is the prediction of real-valued target variables and model identification using genetic programming (GP) algorithms. In data mining, prediction and model identification refer to what is known in control engineering as system identification, or in statistics as curve fitting and regression. The goal is to establish the relationship between input and output (target) variables. In all regards it is a centuries-old "black box" approach, but with one major distinction – the structure of input-output mapping is not fixed, but rather learned from the data. Thus, model induction is both structural and parametric in nature, since both functional form and parameters are adjusted simultaneously. This approach is well known in the neural network community, e.g. as cascade-correlation networks [33], and as structural learning supported by evolutionary search [5]. At the same time, genetic programming [75] has revolutionized the way in which the models are built in many application domains. Genetic programming is capable of operating with arbitrary functional components and carring out non-random data-driven search. Numerous volumes [76][77] document the exceptional ability of GP in finding highly non-trivial and effective solutions to a wide variety of problems.

However, GP also has very high computational requirements. GP search often evaluates tens of millions of candidate solutions against given sample data. Even though

GP excels in obtaining good model structures, it also has other unwanted characteris such as a relatively poor ability to find the numerical values of parameters. In this the we offer a new approach that allows significant improvement in the accuracy of lear models as well as in speed-up of the GP learning process. The proposed approac based on the algorithmic differentiation of GP programs during runs. Algorith differentiation achieves very high precision for parametric values, which are 'ephem random constants' in GP terminology. The associated computational cost is relatively and is compensated by significant gains from overall reduction in training time. A gen framework for tree-structured models is introduced. An analysis of data fitting accur learning dynamics and complexity of the resulting structures is performed. We relate approach with other hybrid "memetic" algorithms [111] utilizing both global and lo learning in either Lamarckian or Baldwinian paradigm [134].

The *third main part* of the work focuses on clustering. While numerous cluste algorithms are available, they tend to be either non-generic or not very robust. This asp of the research contributes to a new area of clustering analysis, namely, the fusion of results from multiple clustering algorithms. In contrast to supervised classificat clustering is an inherently ill-posed problem, whose solution violates at least one of common assumptions about either scale-invariance, richness, or cluster consistency [Different clustering solutions may seem equally plausible without a priori knowle about the underlying data distributions. The exploratory nature of clustering ta demands efficient methods that would benefit from combining the strengths of m individual clustering algorithms. This is the focus of research on clustering ensemb seeking a combination of multiple partitions that provides improved overall clustering

the given data. Clustering ensembles can go beyond what is typically achieved by single clustering algorithm in several respects:

- *Robustness*. Better average performance across the domains and datasets.
- *Novelty*. Finding a combined solution unattainable by any single clustering algorithm
- *Stability and confidence estimation*. Clustering solutions with lower sensitivity noise, outliers or sampling variations. Clustering uncertainty can be assessed fr ensemble distributions.
- *Parallelization and Scalability*. Parallel clustering of data subsets with subseque combination of results. Ability to integrate solutions from multiple distributed sourc of data or attributes (features).

A data set can be clustered in many ways, depending on the algorithm employ and/or its initialization and parameter settings. Can multiple clusterings be combined that the final partitioning of data improves clustering accuracy? The answer depends the quality of elementary constituents of the combination as well as particulars of fusion method. This study departs from a traditional combination approach in seve respects.

First, we develop and study combinations of partitions produced by weak a extremely weak clustering algorithms that use data projections and random cu Traditional methods rely on clustering algorithms that are powerful on their own, and such are computationally involved. We argue that it is possible to generate the partitio using weak, but less expensive, clustering algorithms and still achieve comparable better performance. Certainly, the key motivation is that the synergy of many su components will compensate for any individual weaknesses.

Second, we study two clustering combination methods: (i) using a generalized mutual information criterion, and (ii) based on mixture model estimation via the EM algorithm. As a result, we show that the consensus function can be related to the classical intra-class variance criterion using the generalized entropy and mutual information definition. We propose a probabilistic model of consensus using a finite mixture of multinomial distributions in a space of clusterings. A combined partition is found as a solution to the corresponding maximum likelihood problem using the EM algorithm. We also analyze clustering ensembles with incomplete information and the effect of missing cluster labels on the quality of overall consensus.

Third, we propose and evaluate a framework of adaptive clustering ensembles which generate the partitions using dynamically and non-independently sampled data points. Inspired by the success of supervised boosting algorithms, we devise an adaptive scheme for integration of multiple non-independent clusterings. Individual partitions in the ensemble are sequentially generated by clustering specially selected subsamples of the given data set. The sampling probability for each data point dynamically depends on the consistency of its previous assignments in the ensemble. New subsamples are drawn to increasingly focus on the problematic regions of the input feature space. A measure of a data point's clustering consistency is defined to guide this adaptation.

We analyze clustering combination accuracy as a function of parameters, which control the power and resolution of elementary partitions as well as the learning dynamics versus the number of clusterings involved. A simple explanatory model is offered for behavior of a combination of one-dimensional projection clusterings and combination of partitions by random cuts. We also introduce a unified representation for

combining multiple clusterings and formulate the corresponding categorical clustering problem. A weak clustering is defined via a mutual information distance metric.

Chapter 2

Background on dimensionality reduction

Data mining is a science of discovering knowledge in the form of non-superfluous relationships and patterns in gathered data. Effectively it amounts to building a model of target variables based on other variables. This modeling is often formulated as a prediction, regression or classification problem. The complexity of the data can be enormous, with millions of entries and thousands of dimensions or parameters, both continuous and categorical. The results can be represented in many ways. The model could consist of explicit rules or be a black box model – e.g., a neural network. In either case the usefulness of results strongly depends on the features in use. Potential benefits of reducing the data dimensions include:

1. Improving the modeling (classification/prediction) accuracy.
2. Removing redundant and non-informative features.
3. Simplification of the developed model.
4. Learning (training) becomes easier and less time consuming.
5. Lower measurements cost.
6. Reliability of the parameter estimation improves.
7. Modeling and inference are faster with fewer parameters

Use of unnecessary features may not be harmless for classification accuracy for several reasons. First, some features do not have any information about the classes of interest. This happens if the joint probability density of feature variable $\boldsymbol{x}$ and class label variable $\boldsymbol{\omega}$ factorizes:

$$p_{x\omega}(\mathbf{x},\boldsymbol{\omega}) \rightarrow p_x(\mathbf{x})p_{\omega}(\boldsymbol{\omega})$$

Observation of variable $\mathbf{x}$ does not help to infer the value of $\boldsymbol{\omega}$ because they are statistically independent. Second, the accuracy of some classifiers may suffer if some input features $\mathbf{x}$ and $\mathbf{y}$ are highly correlated:

$$|Corr(\mathbf{x},\mathbf{y})| \rightarrow 1$$

Any classification method relying on feature independence, e.g. maximum likelihood assignment or naïve (simple) Bayes classifier, can potentially perform worse when both $\mathbf{x}$ and $\mathbf{y}$ are included.

Let us review important dimensionality reduction algorithms to provide a background for our own developments in the field. The terminology and notation used follows classical texts such as [26][64][28]. We mostly use the term feature to denote the component of a measurement vector, as commonly accepted in statistical pattern recognition. Attribute, variable, parameter and field are other equivalent terms and can be used in context of a specific application. Feature *selection* algorithms are concerned with finding the best subset among existing features according to some criterion. Feature *extraction* algorithms usually synthesize a set of "new" features. Suppose there are D measurements representing the pattern and Y is a complete set of features:

$$Y = \{y_i \mid i = 1,2,...,D\}$$

Feature selection is concerned with finding the subset X that maximizes chosen criterion function $J(\cdot)$:

$$J(X) = \max_{Z \subseteq Y} J(Z)$$

$$X = \{x_i \mid i = 1, 2, ..., d\}$$

The number of features d in subset X can be fixed in advance or found during the search. Feature selection either accepts a feature as is, or not at all. In contrast, feature extraction transforms existing variables Y into the new variables X:

$$X = W(Y)$$

The dimensionality of the transformed set X is generally lower than that of the original feature space. Feature extraction transformation $W(Y)$ is chosen to optimize an objective function $J(\cdot)$:

$$J(W(Y)) = \max_{W'} J(W'(Y))$$

Another taxonomy of methods is based on the types of criteria used for optimization. In one category, the dimensionality reduction is performed independently of the subsequent classification. The relevance of candidate feature set or transformation is estimated *without* running the actual model but by analyzing data distribution, often heuristically. Such methods are called filters. Filtering itself can rely on other models, e.g. decision trees or neural networks [87]. The other category is the *wrapper* methods – the optimization criterion is the prediction accuracy achieved by the selected (extracted) features. The very same induction algorithm, intended for use after the optimization, produces the accuracy estimate. Both categories have they advantages discussed in more detail below.

The complexity of the feature selection problem can very high. The total number of possible subsets with d features out of D is $\binom{D}{d}$. In general, all possible feature combinations should be evaluated in order to guarantee the optimal subset of features, except certain problems with special structure or size. Moreover, the optimality is achieved only with respect to a chosen evaluation criterion. It is important to note, that a new solution may be required if the evaluation criterion is changed, e.g., if naïve Bayes classifier is replaced by k-NN classification, etc. As pointed out in [26], feature selection or extraction and classifier design are *not* independent, and an optimal subset of features is simply a part of the best classifier, at least conceptually. Search strategy can be considered independent of model, but it efficiency can be vastly improved if domain specific knowledge is taken into account.

Brute force evaluation of all the subsets is not practical when the number of features is large. Heuristic search finds sub-optimal solutions in subset space. Still it is possible to obtain optimal solutions in certain problems, without exhaustive search, because clever search organization can exclude most of the search space states without their explicit evaluation [82]. Hence there are two search strategies: optimal and sub-optimal.

Thus summarizing the taxonomy, feature selection algorithms are usually defined by following components:

- *Evaluation criterion.* Filter-type algorithms do not use a classifier or induction step for selecting subsets. Instead, filtering evaluates features based only on such data properties as distance, information or consistency. Filter algorithms serve as

preprocessors for further induction steps. Wrapper-type algorithms evaluate features based on performance of the model or the classifier learned.

- *Search strategies.* Optimal algorithms always find the best subset of features. In worst case, they require $O(2^d T)$ total computational effort, if T is required for each subset. Heuristic algorithms do not guarantee optimality, but are often the only feasible approach to practical problems.
- *Initialization and Termination.* Initialization sets the starting point for search. In single point algorithms, it can be the empty subset or a set with all the features included, or even some random subset of features. Multipoint (population) search starts with many candidate solutions. Termination is usually controlled by computational effort or quality of solution.

2.1 Optimal subset search

The earliest non-exhaustive yet complete search that finds optimal subsets is a branch and bound feature selection due to Narendra and Fukunaga [96]. It is based on the assumption that the evaluation criterion is monotonic; that is, for the nested feature sets $\overline{\xi}_k$,

$$\overline{\xi}_1 \supset \overline{\xi}_2 \supset \ldots \supset \overline{\xi}_i,$$

the criterion function fulfills the property:

$$J(\overline{\xi}_1) \supset J(\overline{\xi}_2) \supset \ldots \supset J(\overline{\xi}_i),$$

The monotonicity condition allows the algorithm to omit examination of entire branches in a search tree and effectively reduce the computational effort. The basic scheme of Narendra and Fukunaga [96] was further significantly improved in [146] [121] and made

several times faster. Unfortunately, the monotonic criterion is rarely available in practical data mining tasks.

The FOCUS algorithm by Almuallin and Dietterich [3][4] offers optimal search for finding the best model explaining all the training instances. The search is optimal in the sense that it outputs a concept consistent with the training data. However, it works only with binary features. Its computational complexity is bounded by $O(n(2d)^{\log(s-d)})$. The main idea of FOCUS was expanded by Schlimmer [117] with an efficient implementation of the pruning heuristics for finding optimal solutions. Breadth-first search is used in both approaches.

The automatic branch and bound algorithm of Liu et al. [86] attempts to construct monotonic evaluation criterion for general data in order to make regular branch and bound applicable. This measure is called "inconsistency" and is computed for every feature subset based on the number of identical (except class labels) instances. Using the inconsistency measure the search becomes complete but not exhaustive.

In practice, researchers often seek satisfactory suboptimal solutions because problem size or non-monotonic criteria prohibit complete search. Most research is concentrated on heuristic algorithms.

2.2 Heuristic search

Heuristic search algorithms aim to provide reasonably good solutions within affordable time. Two important results in the area greatly influence our understanding of feature selection problems:

1. Sequential selection of features cannot guarantee that an optimal subset will be found, as shown in [21]. This is true for any nonexhaustive sequential algorithm.
2. The best individual feature is not necessarily a part of the two best features in combination [20].

Perhaps the simplest approach is to collect all the best individual features. Apparently, such a set can be suboptimal when features are not independent. Only when the evaluation criterion is a sum or product of individual features' functions, can optimality be preserved.

Several simple strategies employ two basic operations – addition and deletion – of features in a subset [73]. Sequential forward selection adds (on each step) one feature to an initially empty set until no improvement is possible. The feature is retained if the criterion function is maximized by a subset. However, a feature cannot be discarded later. Backward elimination starts with the set of all the features. At each step one feature is deleted that maximizes evaluation value of the remaining features. In general, various heuristics can be constructed by adopting of one of many know variants of hill-climbing. For example, "plus l-take away r" methods on each step attempt to add l features and then delete r features [123].

Floating search [106][107] is a modification of basic sequential search. These algorithms do not have preset values of l and r but control them dynamically. Floating methods update current subsets by both forward and backward selection if the action results in better performance.

Sequential methods consider one solution at a time. In contrast, evolutionary algorithms (EAs) operate on a population of candidate solutions. In a genetic algorithm

(GA) each solution is typically encoded in a binary vector of length d. Subsets of features are represented by 1's in the associated vector positions. Various crossover and mutation operators support search through subsets over many generations. The GA approach to feature selection was pioneered by Siedlecki and Sklansky [118]. GA search performs well on large-scale problems, where it quick identifies promising feature subsets. It could be further combined with hill-climbing to allow fine-tuning of candidate solutions. GAs are certainly not sensitive to non-monotonicity of objective function and not as prone as other methods to difficulties of finding hidden nested sets.

2.3 Filter methods

The evaluation criterion plays a key role in feature selection, since it strongly influences the overall search complexity as well as generality of obtained results. It is rarely completely dictated by the application. Preprocessing of features, before actual model induction, can be done by filter methods. Filtering selects features by analyzing the data. Interclass distances, consistency, information theoretic, and even auxiliary learning algorithms often used as filter measures for feature selection and extraction.

Interclass distances can be estimated from data instances according to several metrics. The search goal is to find a feature subset that provides maximum separation between classes. Classical metrics are reviewed in detail in [26]. They fall into two major categories: 1) deterministic, those that explicitly compute distances between instances (points) of classes, and 2) probabilistic, those that compute distances between class-conditional probability distributions of points. The first category includes such well-

known metrics as Euclidean, Minkowski, Chebychev, city block, quadratic, etc. Probabilistic measures include mutual information, Chernoff, Bhattacharaya, Joshi, Kullback-Leibler divergence, etc. Complexity of computing distances using probability metrics is comparable to estimating error probability. In this case error probability criterion is recommended. The main advantage of filter methods is their speed, since classifier induction is not the part of the search algorithm.

2.4 Wrapper methods

Wrapper methods have access not only to the data but also to the classifier itself during the search. The performance of a candidate solution is estimated during the search by running a classification algorithm using the candidate feature's subset. It usually results in a tighter fit between the classifier and features selected. Fig. 2.1 shows overall wrapper construction with a classifier within the search loop. A number of methods could be used to estimate the performance of the classifier. An important goal of any classification procedure is to achieve the highest possible accuracy with respect to previously unseen individuals. This would mean access to the unknown individuals to evaluate such accuracy, which we will call true or predictive accuracy. In practice we have only a limited number of individuals in a baseline sample.

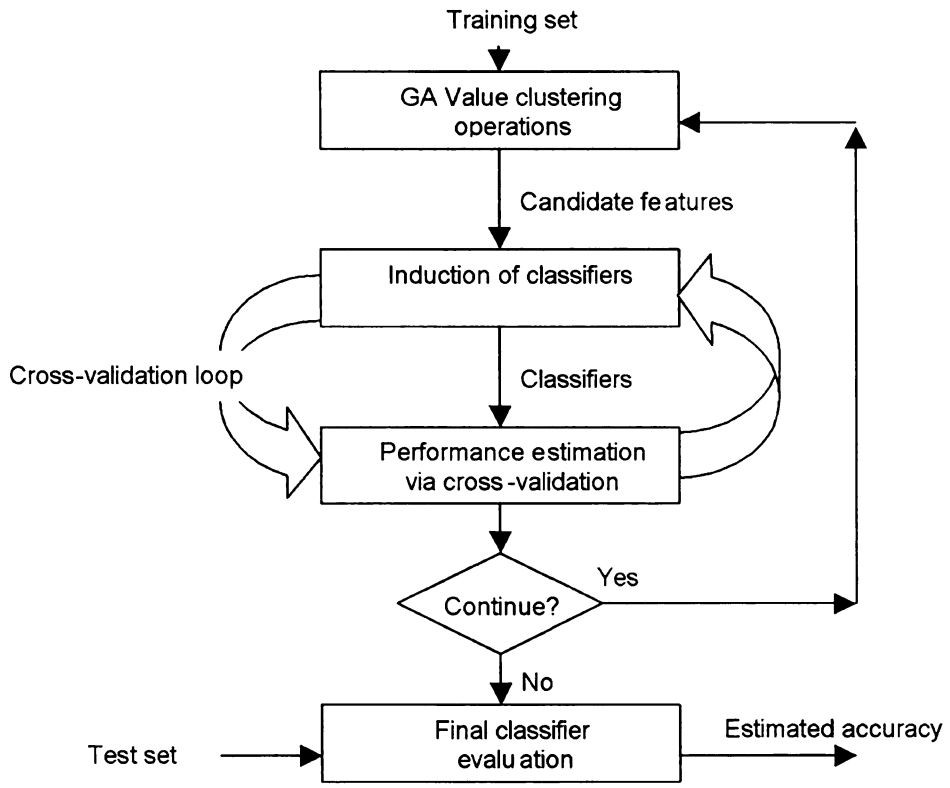


Figure 2.1. Feature extraction wrapper-type approach with GA search engine

Use of all the individuals from the baseline results in a better classifier as opposed to a classifier built only from a subsample of individuals. At the same time, it is necessary to have an accurate measure of classification accuracy even in the absence of independent individuals. Several measures are available that can approximate true accuracy: cross-validation accuracy, bootstrapping [23] and leave-one-out accuracy as a specific case of cross-validation. Leave-one-out (jackknife) evaluation of a classifier is most commonly used, since it is virtually unbiased. Absence of bias means that on average (over different samples) we accurately estimate the true accuracy. A jackknife accuracy estimate is a mean of accuracies of N classifiers, where each classifier is obtained from the original

sample (of size N) by removing a single individual. Each time the (N-1) individuals are used for training and the remaining individual for testing. However, leave-one-out estimates have quite a large variance since estimated accuracy can vary widely for different baseline samples. M-fold cross-validation estimates true accuracy via an average of many classifiers. Each classifier is constructed using [N/M] disjoint subsets of the original sample for testing and the rest of the data for training. Similarly, bootstrap estimates of the true classification accuracy are computed as an average over many bootstrap samples of the original baseline sample. Cross-validation and bootstrapping may provide a better estimation of accuracy since they use more subsamples. However, this increase in complexity gives little improvement in precision in practice. Most researchers consider a jackknife accuracy estimator as the best available or, at least, sufficient given practical constraints.

Jackknife estimation is regularly used in feature selection problems. However, it is often overlooked that leave-one-out estimation itself may mislead feature selection algorithms. In other words, if the search procedure employs a jackknife criterion to sift through various subsets, this may result in a classifier that has a lower true predictive accuracy but better jackknife accuracy. More often than not the selected subset has a lower true accuracy when the jackknife criterion is used with smaller size samples. The main reason for this is that search among classifiers finds a classifier that is overfitting. That is, the classifier fits to the training data well but does not generalize. Jackknife accuracy changes widely as the features vary. Naturally, the features subset having the highest apparent accuracy will be picked during the search, but that does not guarantee better predictive accuracy. Indeed, to prefer one classifier to another, we must make sure

that confidence intervals for their true predictive accuracy do not overlap at the required confidence level. Unfortunately, for small sample size, the confidence intervals are not tight enough and often overlap. As a result the search becomes ineffective. This *over-searching* phenomenon is effectively equivalent to *overtraining*, which is well known for classification algorithms with many degrees of freedom (adjustable parameters), e.g., neural networks.

2.5 Evolutionary feature selection

Evolutionary computing has shown itself to be a highly robust approach to solving difficult optimization problems of structural or numeric nature. The current renaissance of the whole family of evolutionary algorithms (EAs) is due to the new and improved understanding of both strengths and limitations of the evolutionary paradigm [148]. New findings in the theory are tightly coupled with progress in practical applications of evolutionary computations. The No Free Lunch (NFL) theorem by Wolpert and Macready [143] indicates that no single search algorithm is better than the other algorithms for all problems on average. This also means that we can hope to design only algorithms that perform well for certain problems instead of universally good algorithms. It is generally acknowledged that evolutionary search must be careful tailored for the objective function [32]. Still, little is known about how we select the best algorithm for practical problems. Most theoretical results are derived for "toy" problems which are far less challenging than real-world optimization. Optimization practitioners continue to be among the strongest supporters of evolutionary algorithms. While few artificial

benchmarks demonstrate the superiority of standard EAs with respect to conventional optimization, many outstanding results have been obtained by EAs in practice. Every successful EA takes into account distinctive characteristics of real-world problems including:

- Strong non-linear interaction between the parameters of the problem.
- The evaluation function is non-differentiable. Only the value of the objective function is available, but not the derivatives.
- Many constraints exist for the problem (design) variables.
- Scalability of the search algorithm is very important, since search space of the problem can increase in the future.
- Parallel evaluation of several configurations is possible.
- Multiple design criteria may be used simultaneously, thus requiring to consider multiobjective search.
- Robustness of the search algorithm with respect to changes of problem variables.
- High computational complexity. Single objective function evaluation often may take minutes.

Evolutionary algorithms, in particular genetic algorithms (GAs), are a unique kind of optimization method as applied to feature selection/extraction. Unlike conventional sequential search of feature subsets [26], a GA can overcome non-monotonicity of an evaluation function and the difficulties involved in finding hidden, nested sets of features [64]. Siedlecki and Sklansky [118] introduced conventional, direct representation of feature sets. More sophisticated schemes were proposed in [135][136][142]. Building on this seminal work [118], GAs were also applied to feature extraction coupled with *k*-

nearest neighbor classifiers in [108][112][113] and several other algorithms [85][88]. Typically new features are created through the learning of real-value weights applied to the original features [112] or through a sequence of primitive operators encoded in the chromosome [14]. Recent attribute construction algorithms successfully utilize GA for data mining tasks [81][107]. A variety of relevant algorithms is reviewed in [88].

In the next chapter we introduce a different GA approach to dimensionality reduction that does not explicitly operate with whole features, but instead operates on individual values assumed by the feature variables. Features are extracted by clustering several "old" values into a new meta-value which substitutes for the old values in the feature vector. Therefore, new features are created by clustering the values of variables. A GA is used as a search engine for this value clustering. If features assume nominal values then clustering could be viewed as a grouping problem and there are numerous known genetic algorithms that can be used [32].

Grouping and clustering GAs constitute an important research area. Scalability and redundancy are two characteristic problems of grouping algorithms. Here we present a grouping GA with an improved graph-based encoding that extends the work of Park and Song [102]. The proposed cluster representation has lower redundancy while preserving the simplicity of decoding and evaluation. This encoding is not limited to value clustering and therefore applicable to other grouping tasks.

Chapter 3

Genetic Value Clustering

Data mining applications typically deal with instances represented as a set of input attributes (features), where each attribute assumes one of several possible values. We are primarily interested in data represented by nominal values. Throughout the text, the term "value clustering" refers to a method that replaces several original values of an attribute by a new meta-value. This meta-value is completely artificial and simply a substitute for all the values used in the cluster. Such a clustering affects the data model and consequently the classification rule. This clustering of values represents a kind of generalization or value abstraction that is used to improve accuracy of classification. We consider important related work below. However, it is instructive to start with a simple example illustrating the idea of value clustering.

3.1 Clustering in Naïve Bayes Model

Consider a naïve Bayes classifier. The classifier learns the conditional probability $P(A_i|C)$ of each attribute A_i given the class label C. Classification is then done by applying Bayes

rule to compute the probability of C given the particular instance of $A_1,..,A_n$ and then predicting the class with the highest posterior probability (MAP hypothesis):

$$MAP \equiv \arg\max_i P(c_i \mid a_1,..,a_n),$$

where c_i is an instance of C (a class) and a_j is an instance of A_j. There is one major assumption behind this: all the attributes are conditionally independent given the value of the class C. Therefore, by using Bayes rule:

$$P(c_i \mid a_1,..,a_n) = \frac{P(a_1,..,a_n,c_i)}{P(a_1,..,a_n)} \propto P(c_i)\prod_j P(a_j \mid c_i)$$

Let us assume a multinomial probability distribution for the values of each attribute. Using the maximum likelihood principle one can estimate the class-conditional probabilities $\{P(a_j|c_i)\}$ from training samples as a ratio of number of instances with a specific value of a_j within class c_i to the total number of instances in the same class. For example, if there are only three possible values of the first attribute $a_1 \in \{\alpha,\beta,\gamma\}$, one would estimate the three terms $P(a_1=\alpha|c_i)$, $P(a_1=\beta|c_i)$ and $P(a_1=\gamma|c_i)$. However, if values α and β are merged into one cluster, the meta-value x is constructed $x=(\alpha \textit{ or } \beta)$, and only two model parameters remain to be estimated: $P(a_1=x|c_i)$ and $P(a_1=\gamma|c_i)$. In the multinomial model, the probability of the meta-value x obeys:

$$P(a_1=x \mid c_i) = P(a_1=\alpha \mid c_i) + P(a_1=\beta \mid c_i).$$

As a result of this value clustering all the occurrences of α or β are replaced by x. Of course, the estimation of parameters is classifier specific, and can take a different form if another data model is assumed.

3.2 Background on Value Abstraction and Clustering

In traditional database research there is existing literature on the methods of finding an approximate answer for a database query with no exact answer [92]. Automatic query expansion systems offer approximations from a cluster of similar values, obtained by term clustering [122]. Term clustering creates groups of related words by analyzing their co-occurrence in a document collection. A hierarchy of values can either be garnered from experts or obtained automatically. In the work [89], the attribute values are clustered automatically by generating an attribute abstraction hierarchy. This hierarchy is discovered using rules derived from database instances. For rules with the same consequence, values found in the premise are clustered. This process is called pattern based knowledge induction. However, information retrieval approaches cannot be easily combined with arbitrary classifiers.

In machine learning studies we find a number of flexible value clustering methods. Perhaps the most interesting method is the information bottleneck by Tishby et al. [126]. The information bottleneck method replaces the original random variable X by a compact representation $\tilde{X}$, which tries to keep as much information as possible about the random variable Y. In particular, variable x could stand for feature values and variable Y is a class label. The information bottleneck method maximizes mutual information $I(\tilde{X};Y)$ between $\tilde{X}$ and Y, conditioned on the information content $I(\tilde{X};X)$ resulting from the clustering $\tilde{X}$ with respect to X. Most significantly, the optimal solution of this information theoretic problem can be found in terms of probability

distributions $p(\tilde{x} \mid x)$, $p(y \mid \tilde{x})$, and $p(\tilde{x})$ with an arbitrary joint distribution $p(x, y)$. The exact solution is given by a set of nonlinear equations [126]. An iterative solution requires the user-defined cardinality of $\tilde{X}$. Unknown variables are usually initialized to random values that potentially may lead to sub-optimal local solutions. The resulting clustering is not "hard", where each value of X belongs to a single cluster in $\tilde{x}$, but rather "soft" with membership probabilities $p(\tilde{x} \mid x)$. However, hard clustering can be achieved in the agglomerative information bottleneck method by Slonim and Tishby [119]. The agglomerative information bottleneck method is a hierarchical process that uses distance measures between distributions and greedy search by merging clusters pair-wise. The multivariate information bottleneck method [45][120] further extends the approach for multivariate distributions, maximizing mutual information between Y and several "bottlenecked" variables simultaneously.

Recently, the value abstraction approach (clustering in our terms) led to considerable progress in linkage analysis for genetic mapping [98], and was expanded for more general likelihood computations and faster Bayesian network inference in [44]. All these works can be characterized as filter-type feature extraction, since classifier accuracy or induction step feedback is not used for value clustering.

The other relevant body of literature concerns the use of genetic algorithms for clustering problems, and specifically for *combinatorial optimization* involving clustering. Combinatorial grouping problems are rarely supplied with a definition of distance between items. Advanced GA designs utilize graph-based solution encodings of the clustering problems. Originally, a graph-based encoding was introduced by Park and

Song [102]. The main motivation of their study was to re-solve the problems of traditional grouping encodings that use group numbers or permutations with separators.

In the simplest case a chromosome contains a group label (number) for each object. Such a group-number encoding suffers from the incompatibility of labels between two chromosomes that ultimately participatie in genetic recombination. In fact, by using group numbers we are forced to solve an intractable correspondence problem between different labelings each time a crossover operator is applied. In group-number representation typical crossover cannot easily combine objects' memberships from parents to offspring. In addition, the number of groups must be known in advance. Moreover, this encoding contains redundancy because it does not take into account the symmetry of group label permutations. The GA community currently rejects such a representation in its raw form [32]. However, some heuristics can certainly help in adopting group-number representation [83][19], but their complexity leads to major computation overhead to ensure meaningful inheritance.

Another popular encoding uses a permutation representation [32]. A chromosome is created by an arbitrary permutation of the original objects. There is no cluster information in the chromosome itself. Instead, the responsibility of constructing a clustering from the chromosome lies on a special decoder function, which is problem specific. The decoder function is guided by the particular optimization criterion for the given task. The drawback of the permutation representation is basically the same as that of the group-number encoding: classical recombination of two chromosomes cannot combine two clusterings by transferring cluster content, since clustering of the offspring

is exposed only by a decoder function. Membership of each individual item acquires its meaning only in the context of labeling of other items.

More effective design of a grouping genetic algorithm (GGA) is due to Falkenauer [31][32]; it operates with a permutation encoding with separators between groups. GGA requires sophisticated repairs during the crossover to ensure valid offspring representation. In GGA, the crossover operator must be domain dependent in order to enable effective propagation of good building blocks (clusters) to the next generation.

3.3 Cluster Representation in Genetic Algorithm

We propose a novel genetic algorithm design for general clustering, and value clustering in particular. Value clustering can be regarded as a grouping problem, where a number of objects (items) must be placed in several groups. Examples of grouping problems include bin packing, graph coloring and line balancing, all of which are NP-hard. The search space size for partitioning N objects into exactly M groups grows exponentially with N. Stirling numbers of the second kind $S(N, M)$ give the number of ways to partition N objects into M groups:

$$S(N,M) = \frac{1}{M!}\sum_{i=1}^{M}(-1)^{M-i}\binom{M}{i}i^{N}$$

If the number of clusters is not known then M can assume any value between 1 and N. Thus we get the total number of distinct clusterings, also known as the Bell number $B(N)$:

$$B(N) = \sum_{M=1}^{N} S(N,M)$$

For example, for N=1...10 we obtain Bell numbers $B(N)$ = {1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975}.

The major difficulty encountered by typical GA approaches is in designing a representation of a grouping as a chromosome. A good encoding must ensure the complete coverage of the space of possible partitions with minimal redundancy. This encoding should also allow meaningful inheritance of information from parents to children. Finally, it is important that validity checks and chromosome repair have insignificant computational overhead, if in fact such repair is necessary.

Several genetic algorithms are specifically known to solve grouping problems. The most straightforward approach uses a standard GA with group-numbers, one gene per object, where each gene contains a group-number for the object. Unfortunately, standard n-point crossover fails to transmit grouping information between chromosomes, since group-numbers do not carry meaning by themselves and could be arbitrarily permuted for any given grouping. This representation also requires knowing the correct number of clusters in advance. Similar difficulties are encountered with permutation encodings. In contrast, the GGA algorithm [31] guarantees meaningful inheritance using permutation with separators and a sophisticated crossover operator with repairs.

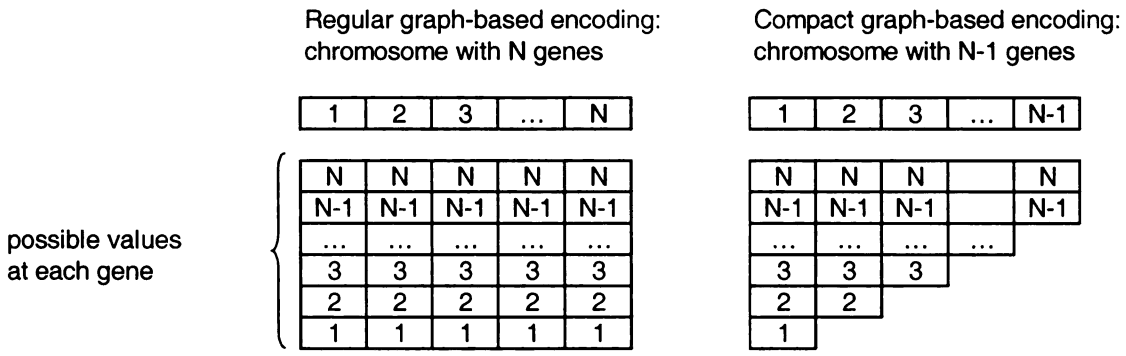


Figure 3.1. Regular graph-based clustering GA selects N genes from the total of N^2 edges, while compact encoding needs only N-1 from N (N-1)/2 edges. GA search space size is reduced from N^N to $N!$.

Graph-based encoding is a different method for solving clustering problems [102]. If there are N items to cluster then a solution is represented by a string with N genes, one gene per item. Given that original items are numbered from 1 to N, then each gene assumes a value between 1 and N. Thus if the i-th gene contains value j, then items with numbers i and j belong to the same cluster and can be connected by a link in a graph of all items. In this representation an arbitrary partitioning can be reached without pre-defining a correct number of clusters. No special crossover is required for this representation, since even standard crossover both transfers the item's link and preserves information about the partitioning. However, the redundancy of the encoding is extremely large, since there are N^N possible individuals. Many points in the GA search space correspond to the same cluster partitioning, because a cluster can be formed by any connected graph containing its items. The redundancy of the regular graph-based encoding is given by the ratio of GA search space size to Bell number $B(N)$:

$$\rho(N) = \frac{N^N}{B(N)}.$$

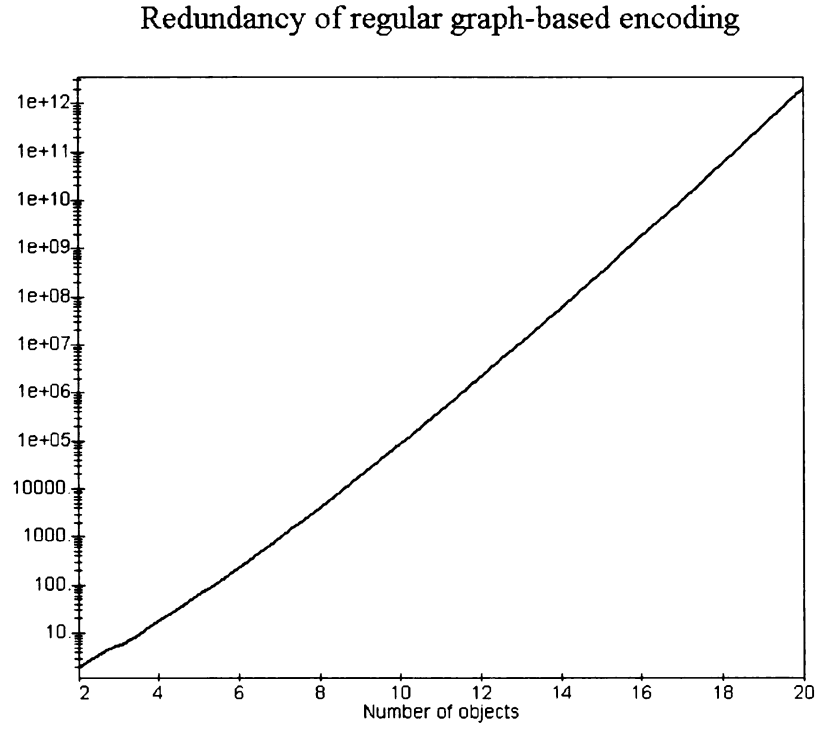


Figure 3.2. Ratio of GA search space size to the true number of distinct partitions of objects on logarithmic scale.

For example, for 20 objects we find $\rho(20) > 2 \cdot 10^{12}$.

Park and Song have proposed a remedy to this redundancy problem, by limiting the number of possible links based on domain knowledge [102]. For example, if the distance between objects can be defined, then one may restrict the possible links for this object to a set of nearest neighbors. Unfortunately, feature value clustering as well as the combinatorial grouping problem in general do not allow introduction of meaningful metrics or definitions of neighborhood for nominal values.

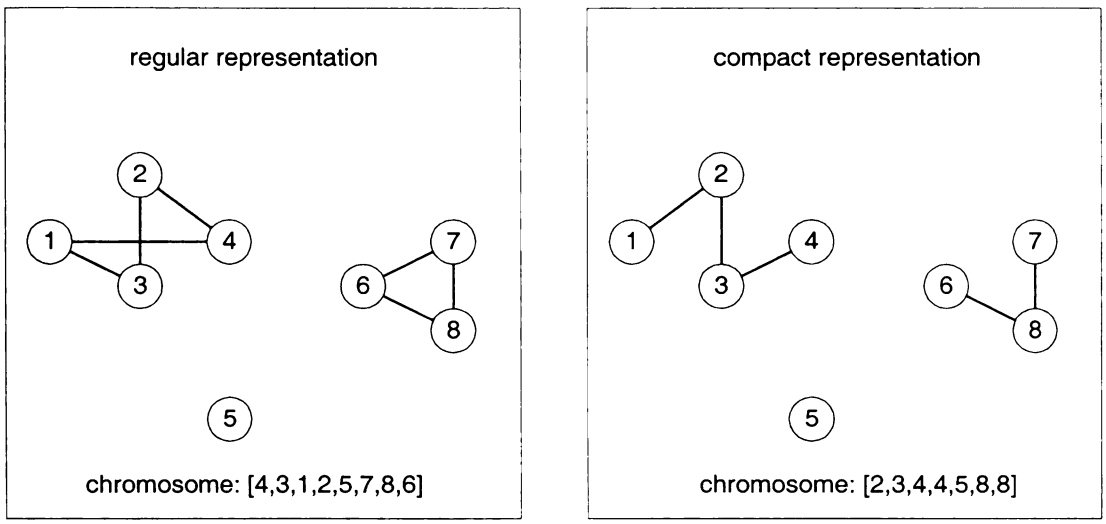


Figure 3.3. Sample chromosomes and corresponding clusterings are shown by connected subgraphs for regular and compact encodings

Here we present an improved compact graph-based encoding, where redundancy is reduced exponentially with respect to number of objects N, as compared to the original idea discussed in [102]. The proposed representation keeps all the good qualities of the raw graph-based encoding, without sacrificing its representation power, as we strictly prove below.

First, we note that the regular graph-based encoding picks N links out of a total N^2 available, because the chromosome has N genes with N possible values per gene. However, even a complete graph on N vertices has only $N(N\text{-}1)/2$ edges. This results in a major combinatorial explosion in the number of ways to encode the same cluster. Furthermore, to represent any partition of items on a graph we need at most N-1 edges. Thus an arbitrary tree connecting all the objects within a cluster would be sufficient.

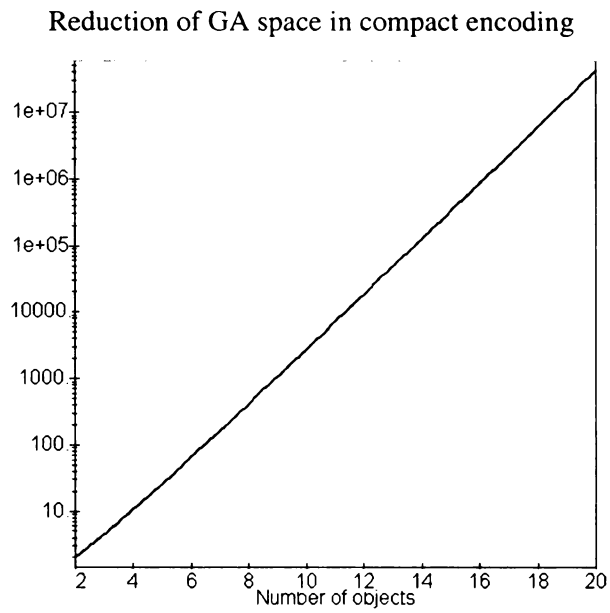


Figure 3.4. Ratio of GA search space sizes in non-compact and compact encoding

Indeed, the largest possible cluster contains N items that could be represented by a tree with N-1 edges.

We propose an enhanced encoding that draws gene values only from $N(N\text{-}1)/2$ edges and needs N-1 genes. The main idea is to remove duplicated edges from a pool of possible gene values. Retaining only values from N to i, for the i-th gene, does this exactly. The first gene assumes values in {1,...,N}, the second gene in {2,...,N}, etc. The N-th gene is unnecessary, since it always is set to N, and can be omitted. If the gene value happens to be equal to the gene's number, it means that no edge is placed on the graph. We cannot have less than $N(N\text{-}1)/2$ edges of a complete graph, because clusters with only two objects must be obtained by a unique link between them. Figure 3.1 compares the regular and compact graph-based encoding. Figure 3.2 shows graph representations of the sample chromosomes.

In the compact encoding, each gene obtains its values from an alphabet of a different cardinality. This raises a question as to how required clusters are formed. It is easy to prove that any possible clustering is still attainable in the new encoding. For this purpose, consider a cluster containing objects with arbitrary numbers. One can sort objects in a cluster by their number. Any such cluster can be formed, by connecting an object with next higher numbered object from the sorted list, because the required edges are always available: any object has access to edges incident to any object with greater number. Therefore, the proposed encoding has the same representational power at the original encoding but has an exponentially lower space, as we approximately estimate from:

$$\delta = \frac{N^N}{N!} \approx \frac{e^N}{\sqrt{2\pi N}},$$

where δ is the ratio of sizes of GA search space in old and new encodings. For example, if N=20 we find that δ>43000000, a dramatic reduction!

The compact encoding has one problem, namely different genes carry different information because of the different cardinalities. This could be an issue for a standard crossover (e.g. 1- or 2-point) since lower numbered positions on the chromosome are more important than the higher numbered positions. As an alternative, we offer a modification of the compact encoding that equalizes the cardinality of alphabets for different genes. The genes with the most available edges can transfer some of their edges to other genes, so we still have the very same pool of $N(N\text{-}1)/2$ edges at our disposal. For

example, we can delete the value *(N-1)* in the alphabet of the first gene, and insert value 1 in the alphabet of (*N*-1)th gene, leaving us with the same edges. Equalization creates alphabet of cardinality |A|=(1+*N*/2) for every gene. Equalization can be performed exactly with even valued *N*, with minor deviation at some positions when *N* is an odd number. In the equalized compact representation, the reduction in size of search space is still greater than before, namely:

$$\delta = \frac{N^N}{(1+N/2)^{N-1}} = \left(1+\frac{N}{2}\right)\left(\frac{2}{1+2/N}\right)^N$$

Let us prove that it is possible to realize any clustering in the equalized compact representation. We will show that a cluster with an arbitrary number of objects can be created. The alphabet of the genes (corresponding to any chosen objects) contains all possible links between these objects, because equalization preserves the edges by simple transfer. Now an auxiliary construction is necessary: imagine a complete graph on these objects. In this complete graph we can assign direction to the edge from object *I* to *J* if the alphabet of gene *I* contains value *J*, otherwise the edge goes in the opposite direction. This directed graph is called a tournament since it is a complete and directed graph. Graph theory provides us with this important fact: every tournament has a Hamiltonian path [17]. A Hamiltonian path connects all the objects following the arcs and visits each and every object only once. Therefore, arcs from such a directed path are equivalent to valid assignment of edges to genes. Since arbitrary objects are connected, we can obtain any partitioning. Figure 3.5 shows an example comparing compact and equalized graph-based encoding for a set of 6 objects.

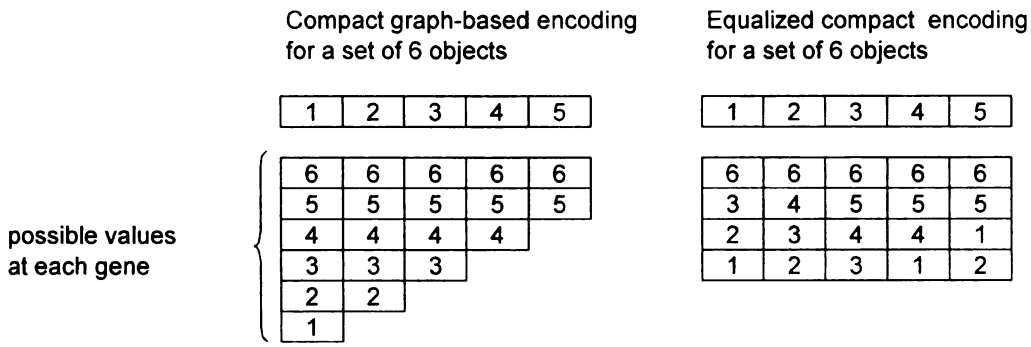


Figure 3.5. An example of encoding alphabets at each chromosome position in compact and equalized compact graph-based representations of a set of 6 items

For most of the experiments we used an equalized compact encoding and 2-point crossover. The next section details several application domains for value clustering.

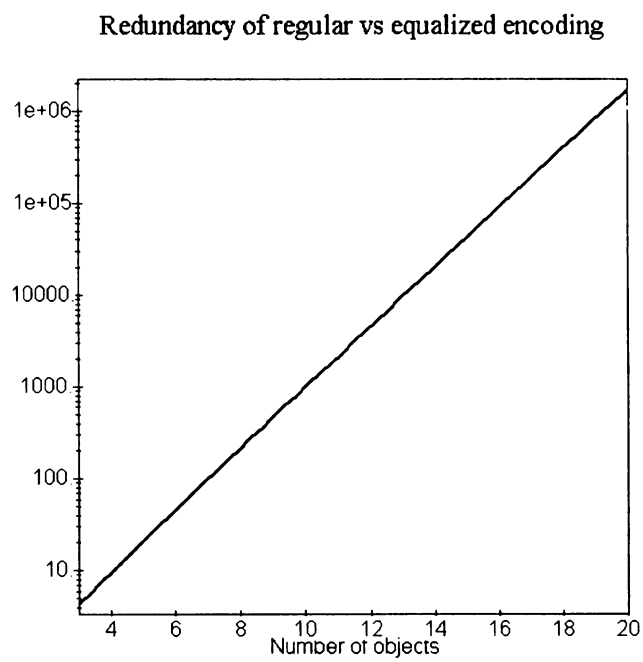


Figure 3.6. Reduction of search space size in equalized compact encoding as compared to regular graph-based encoding.

3.4 Application Domains for Value Clustering

Text categorization

Text categorization attempts to automatically place documents into one of a set of predefined classes. Training samples for the experiments were taken from the Reuters-21578 documents collection. We use a classical "bag of words" data model using a naïve Bayes classifier. The number of features is equal to the number of words in a document and each feature can assume any value from the observed dictionary of size *N*. The dictionary is created from all the training data. The classifier is induced by estimating the probabilities of words in the entire training documents collection. If a previously unseen word appears during the inference phase, its probability is set to a small value such as 0.1/N. Note that the number of features changes from document to document, while the number of feature values is constant for a fixed training set. It is our goal to cluster the values within features. The probability distribution for each feature is the same due to the independence assumption used in naïve Bayes. Therefore, the clustering of feature values is the same for all features. Thus each GA chromosome represents a clustering of words in the dictionary. The full dictionary of the Reuters data set contains tens of thousands of words (values to be clustered). It is commonly acknowledged that preprocessing is necessary to reduce computational costs to a reasonable level. We perform preprocessing by retaining features having the highest mutual information within the category label. Mutual information is assessed independently for each feature. Original baseline documents from each class are split between training and holdout sets (for classifier

fitness evaluation), as well as a final test set to estimate the true accuracy of the best solution found. After the search phase, the best classifier was induced from a combination of training and holdout sets before the final testing. The comparison between the three methods could be made, namely, a naïve non-clustered approach, our proposed clustering GA and a pure best-first hill climbing with the same number of fitness evaluations.

DNA fingerprinting

Assignment of individuals to their putative populations of origin is a practical problem in population genetics. In a classical framework (see, e.g. Waser and Strobeck [139]), to assign an individual it is necessary to compute likelihood functions for each source population. The population having the highest likelihood value is deemed to be the most probable population of origin. Genetic markers, taken from DNA samples, serve as features and their values (allelic configurations) are assumed to be multinomially distributed [53]. Classification is then performed using a naïve Bayes approach, assuming independence of features. The large number of possible alleles (e.g.10-20) for each feature seriously complicates the assignment in many practical situations, because the reliability of the estimated parameters is low, since a typical baseline sample consists of only 50-100 individuals. Value clustering is quite promising for binning the alleles together to improve the assignments.

Bayesian network classifiers

A Bayesian network, also known as belief network, is a directed acyclic graph, such that:

(1) A set of random variables makes up the nodes of the network.

(2) Each directed link in the network represents *direct influence*. An arrow from node *A* to node *B* means that *A* has a *direct influence* on *B*.

(3) Each node has a conditional probability table that quantifies the effects that the parents have on the node.

In theory, a belief network is very powerful because it can represent any joint probability distribution of its node variables. However, in practice a probabilistic model matching the data sets cannot always be found. Belief networks can inference about any probabilistic quantity of interest because theoretically everything can be obtained from the joint distribution. However, in practice, polynomial algorithms may not be available under the condition of some values not being instantiated in the network. Good approximate inference procedures do exist.

Our primary interest is to consider solving classification problems for discrete data analysis. Discrete data is very common in genetics data. The reason that we prefer belief networks is because

1. They can model joint probabilities with arbitrary accuracy, taking into account dependencies.
2. Unlike neural networks, their representation of interactions between the variables can be interpreted and understood.
3. Unlike decision tree or decision rules, they are not very sensitive to noise.

It is true that finding a general exact belief network for a particular set of data is a task of exponential complexity. Therefore, some simplifications are necessary which would limit the space of possible structures. For example, two of types of inductive bias could be

considered to simplify learning: naïve Bayesian classifier (trivial), and tree-augmented naive (TAN) Bayesian network.

Tree-augmented Naïve Bayesian Network structure has the following defining properties:

- Class variable (node) has no parents and each attribute (non-class node) has only one or two parents and one of them has to be a class node.
- Edges representing relations between attributes are included.
- In TAN, graph structure among attribute nodes is a tree.

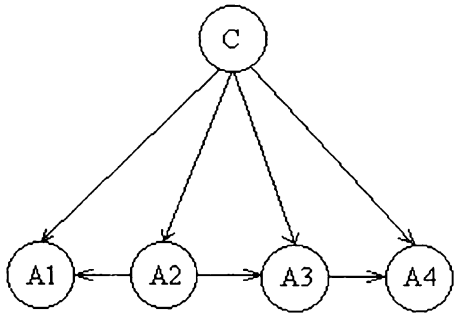


Figure 3.7. TAN structure with four attributes. Augmented edges are shown among the attribute nodes.

$$P(c_i \mid a_1,..,a_4) = \frac{P(a_1,..,a_4,c_i)}{P(a_1,..,a_4)} \propto P(a_1 \mid a_2,c_i)P(a_1 \mid c_i)P(a_3 \mid a_2,c_i)P(a_4 \mid a_3,c_i)$$

The TAN structure is preferred because it exploits dependencies between attributes, which usually provide better accuracy than a naïve Bayesian classifier. Moreover, this structure can be learned in polynomial time. Structural learning can be done in $O(K^2)$ time using the mutual information algorithm [18], where K is number of nodes (variables) in the Bayesian network.

		X			
		1	2	3	4
Y	1	p11	p12	p13	p14
	2	p21	p22	p23	p24
	3	p31	p32	p33	p34
	4	p41	p42	p43	p44

		X		
		1	2	3+4
Y	1	p11	p12	p13+p14
	2+3	p21+p31	p22+p32	p23+p24+p33+p34
	4	p41	p42	p43+p44

Figure 3.8. Conditional probability table P(X,Y|C=c) before value clustering and after the merging of values {3,4} of X variable and values {2, 3} of Y variable

Our goal is to use value clustering for learning conditional probability distributions $P(a_i \mid a_j, c_k)$. Value clustering is viewed as merging various cells in conditional probability tables. Again, it will reduce number of estimated model parameters. Parameters themselves will be more reliable due to the increase in effective sample size. Encoding of TAN node tables in a GA is possible in a number of ways, such as graph-based representation. Value clustering in conditional probability tables can be either in rows , columns, or both simultaneously. For example, if we have P(X,Y|C=c) for X and Y variables, each assuming 4 possible values then initially we need to estimate 16 entries in the table P(X,Y|C=c) for some fixed class. Figure 3.8 shows the result of clustering the values {3,4} of X variable and values {2, 3} of Y variable.

3.5 Closed form analysis of value clustering

In several simple data models we can obtain exact expressions for the effect of value clustering on classifier accuracy. One such data model in feature space is represented by a one-dimensional multinomial variable. In this case one can calculate the change in error rate of classification after clustering of two values, if Bayes classifier parameters are estimated by the maximum-likelihood procedure.

Let X be a random variable that assumes one of M possible values $\{1, 2,\ldots, M\}$ with probabilities $\{p_1, p_2, \ldots, p_M\}$ in class 1, and with probabilities $\{q_1, q_2, \ldots, q_M\}$ in class 2. This is a two-class model, for which we apparently have $p_1+ p_2+ \ldots+ p_M = 1$, $q_1+ q_2+ \ldots+ q_M=1$. Suppose we observe a data sample of size N, where data points are drawn from the two source classes with equal prior probabilities $\pi_1=\pi_2$. Each data point is sampled according to the above class-conditional probabilities:

$$\text{Prob}\ (\ X = i\ \mid 1\ ) = p_i,$$

$$\text{Prob}\ (\ X = i \mid 2\ ) = q_i.$$

Our first step would be to find an optimal decision rule, given the observed finite sample, and estimate its error rate E_{old}. Next we will merge two of the values (characters) from the alphabet of variable X, and estimate the error rate E_{new} of the resulting classifier. And finally we can calculate the change in error rate $\Delta E= E_{\text{old}} - E_{\text{new}}$. Obviously we are ultimately interested in under what distributions of probability values one can obtain the positive value of $\Delta E >0$.

The gain (or loss) of accuracy ΔE is a function of many factors: observed data sample $(x_1, x_2, \ldots, x_N)$, class-conditional probabilities $\{p_1, p_2, \ldots, p_M\}$ and $\{q_1, q_2, \ldots,$

$q_M\}$, and sample size N. However, if we merge the characters 1 and 2, then it can be proved that the value ΔE only depends on the values p_1, p_2, q_1, q_2. To show this we note that for any given data sample D=(x_1, x_2, ..., x_N) the value of error e_D of simple Bayes classifier is an additive function of the parameters of the distribution:

$$e_D = f_1(p_1) + f_2(p_2) + \ldots + f_M(p_M) + g_1(q_1) + g_2(q_2) + \ldots + g_M(q_M),$$

where $\{f_i\}$ and $\{g_i\}$ are certain functions whose exact value is determined by the actual sample drawn. Clearly, when a difference is taken between the error values of classification before and after merging the states 1 and 2 of X, we obtain:

$$e_D(\text{old}) - e_D(\text{new}) = f_1(p_1) + f_2(p_2) + g_1(q_1) + g_2(q_2) - f'_1(p_1 + p_2) + g'_1(q_1 + q_2).$$

Hence, for a given sample the number of relevant parameters is reduced to four -- $\{p_1, p_2\}$and $\{q_1, q_2\}$. At this point we can evaluate the difference of error rates ΔE. Let us recall that the error rate of classifier averaged over the possible training samples can be calculated as the expected value of error on single data sample taken over the probability distribution of data.

$$E[e_D] = \langle e_D \rangle = \int e_D(x_1, x_2, \ldots, x_N)\, p(x_1, x_2, \ldots, x_N)\, dx_1 dx_2 \ldots dx_N$$

For a multinomial samples the distribution is discrete and the integral must be replaced by corresponding sums. For simplicity we will consider training samples with equal number of points N from each class. In the two-class situation, this will mean the total sample size will be $2N=N+N$. The probability of any sample $D=(x_1, x_2, \ldots, x_{2N})$ is then given by the multinomial:

$$p(x_1, x_2, \ldots, x_N) = \frac{N!}{n_1! n_2! \ldots n_M!} p_1^{n_1} p_2^{n_2} \ldots p_M^{n_M} \frac{N!}{m_1! m_2! \ldots m_M!} q_1^{m_1} q_2^{m_2} \ldots q_M^{m_M}$$

where $n_1 + n_2 + \ldots + n_M = N, \quad m_1 + m_2 + \ldots + m_M = N$

Here n_i and m_i denote the number of observations of value $X = i$ in class 1 and class 2, respectively.

The change in error rate is then computed as $\Delta E = E_{old} - E_{new}$ and can be written (after some manipulations) as follows:

$$\Delta E = \frac{1}{2}\sum_{m_2=0}^{N}\sum_{m_1=0}^{N-m_2}\sum_{k_2=0}^{N}\sum_{k_1=0}^{N-k_2}\Bigg(q_1\theta(k_1 - m_1 - 1) + p_1\theta(m_1 - k_1 - 1) + \frac{q_1 + p_1}{2}\delta(m_1 - k_1)$$
$$+ q_2\theta(k_2 - m_2 - 1) + p_2\theta(m_2 - k_2 - 1) + \frac{q_2 + p_2}{2}\delta(m_2 - k_2)$$
$$-(q_1 + q_2)\theta(k_1 + k_2 - m_1 - m_2 - 1) - (p_1 + p_2)\theta(m_1 + m_2 - k_1 - k_2 - 1)$$
$$- \frac{p_1 + p_2 + q_1 + q_2}{2}\delta(k_1 + k_2 - m_1 - m_2)\Bigg) \times$$
$$\times \frac{N!}{k_1!k_2!(N - k_1 - k_2)!} p_1^{k_1} p_2^{k_2} (1 - p_1 - p_2)^{N - k_1 - k_2} \times$$
$$\times \frac{N!}{m_1!m_2!(N - m_1 - m_2)!} q_1^{m_1} q_2^{m_2} (1 - q_1 - q_2)^{N - m_1 - m_2}$$

Where $\delta(z)=1$, if $z=0$, $\delta(z)=0$ otherwise; and $\theta(z)=1$, if $z \geq 0$, $\theta(z)=0$ otherwise.

This expression for ΔE can be evaluated in closed form for particular values of N. Even in the case of fixed N the calculations of such combinatorial sums are quite laborious. For this purpose I used a specialized computer algebra system 'FORM' by Jos Vermaseren [138], initially developed for similar expressions arising from high-energy physics calculations. Certainly, the asymptote of large values of N is less interesting than small sample size, because large sample size usually provides reliable estimates of classifier parameters, thus leaving us with smaller room for improvement due to value clustering.

We computed the change in error rate ΔE for total sample sizes (2N) equal to 10, 20, 30, and 50 for several different relations between the model parameters $\{p_1, p_2\}$ and $\{q_1, q_2\}$. In order to represent the ΔE in tractable form we considered the following four special cases:

$$\{p_1, p_2 = p_1\}, \{q_1, q_2 = q_1\}$$

$$\{p_1, p_2 = 4/5\ p_1\}, \{q_1, q_2 = 5/4\ q_1\}$$

$$\{p_1, p_2 = 2/3\ p_1\}, \{q_1, q_2 = 3/2\ q_1\}$$

$$\{p_1, p_2 = 1/2\ p_1\}, \{q_1, q_2 = 2\ q_1\}$$

These assumptions cover many plausible situations, which reflect gradual increase in discriminative power provided by the values/states {1, 2} selected for clustering from the alphabet of variable X. For instance, configuration $\{p_1, p_2 = p_1\}$ and $\{q_1, q_2 = q_1\}$ corresponds to the case when having distinct values {1, 2} generally does not help in classification. In contrast, when $\{p_1, p_2 = 1/2\ p_1\}$ and $\{q_1, q_2 = 2\ q_1\}$, one can expect considerable contribution to accuracy from both values {1, 2} (not merged), given that the maximum likelihood estimates of the distribution parameters $\{p_1, p_2\}$, $\{q_1, q_2\}$ are known with good precision.

To get an idea of the result of computing the value of gains/losses, consider the following expression obtained for $\{p_1, p_2 = p_1\}$, $\{q_1, q_2 = q_1\}$ and sample size 10 (N=5):

$$\Delta E = E_{old} - E_{new} = 1/2 \cdot (5 - 35700p_1^3q_1^4 - 75p_1^3 - 35700p_1^4q_1^3 + 11430p_1^4q_1^2$$
$$+ 35770p_1^4q_1^4 - 30p_1 - 30q_1 + 70p_1^2 + 31p_1^4 + 43180p_1^3q_1^3 + 8556p_1^2q_1^2$$
$$- 17010p_1^2q_1^3 + 11430p_1^2q_1^4 - 1260p_1q_1^4 + 31q_1^4 + 420p_1q_1 - 75q_1^3 - 1575p_1q_1^2$$
$$+ 2356p_1q_1^3 - 17010p_1^3q_1^2 + 70q_1^2 - 1575p_1^2q_1 - 1260p_1^4q_1 + 2356p_1^3q_1) \cdot (p_1 - q_1)^2$$

The number of terms in other similar results for ΔE would grow as $O(N^4)$, which makes it difficult to interpret and analyze. Instead, we visualize the behavior of gains/losses ΔE in expected error rate on the following plots shown in Figures 3.9-3.14.

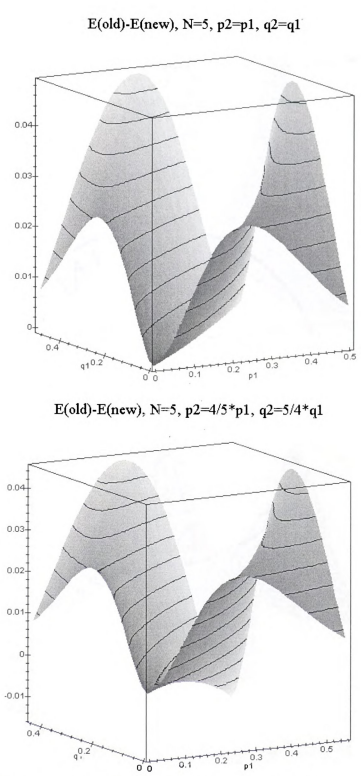
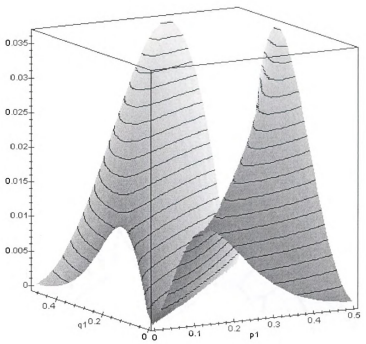


Figure 3.9. Accuracy gain ΔE due to merging states {1,2} for sample size 2N=10: a) if $\{p_1, p_2 = p_1\}$, $\{q_1, q_2 = q_1\}$, b) if $\{p_1, p_2 = 4/5\ p_1\}$, $\{q_1, q_2 = 5/4\ q_1\}$



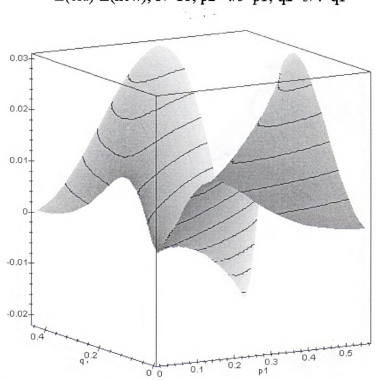
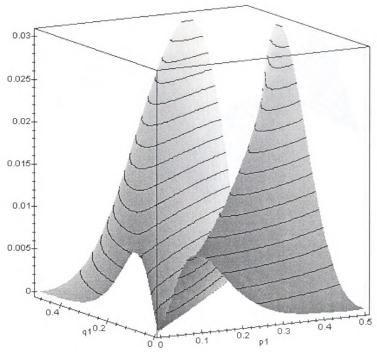


Figure 3.10. Accuracy gain ΔE due to merging states {1,2} for sample size 2N=20: a) if $\{p_1, p_2 = p_1\}$, $\{q_1, q_2 = q_1\}$, b) if $\{p_1, p_2 = 4/5\ p_1\}$, $\{q_1, q_2 = 5/4\ q_1\}$



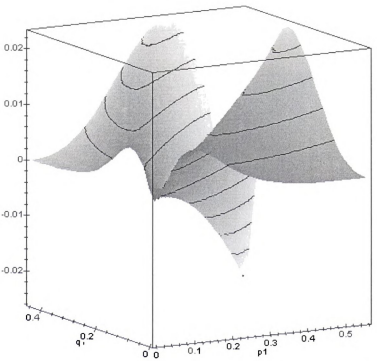
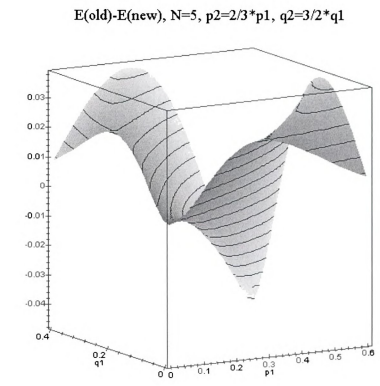


Figure 3.11. Accuracy gain ΔE due to merging states {1,2} for sample size 2N=30: a) if $\{p_1, p_2 = p_1\}$, $\{q_1, q_2 = q_1\}$, b) if $\{p_1, p_2 = 4/5\ p_1\}$, $\{q_1, q_2 = 5/4\ q_1\}$



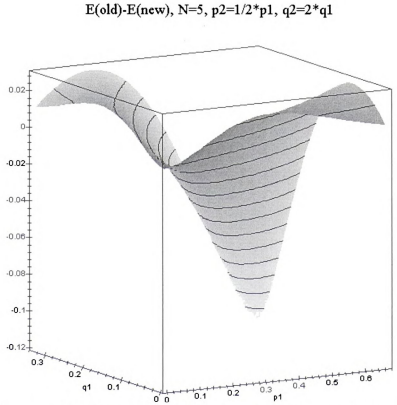
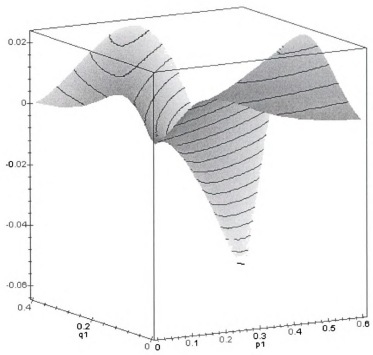


Figure 3.12. Accuracy gain ΔE due to merging states {1,2} for sample size 2N=10: a) $\{p_1, p_2 = 2/3\ p_1\}$, $\{q_1, q_2 = 3/2\ q_1\}$ b) $\{p_1, p_2 = 1/2\ p_1\}$, $\{q_1, q_2 = 2\ q_1\}$



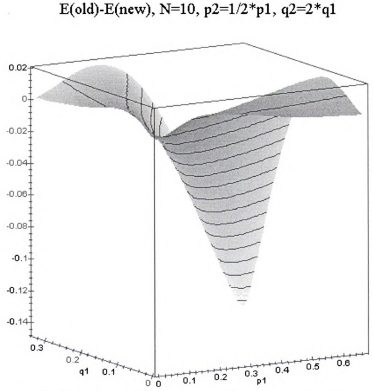
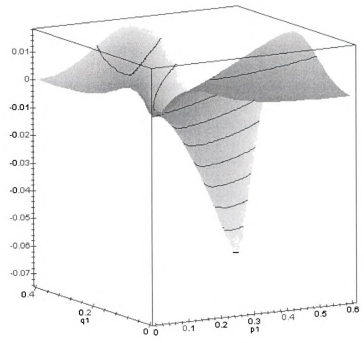


Figure 3.13. Accuracy gain ΔE due to merging states {1,2} for sample size 2N=20: a) $\{p_1, p_2 = 2/3\ p_1\}$, $\{q_1, q_2 = 3/2\ q_1\}$ b) $\{p_1, p_2 = 1/2\ p_1\}$, $\{q_1, q_2 = 2\ q_1\}$



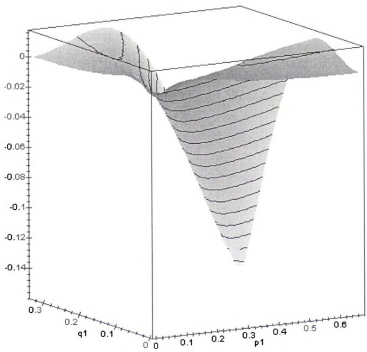


Figure 3.14. Accuracy gain ΔE due to merging states {1,2} for sample size 2N=30:
a) $\{p_1, p_2 = 2/3\ p_1\}$, $\{q_1, q_2 = 3/2\ q_1\}$ b) $\{p_1, p_2 = 1/2\ p_1\}$, $\{q_1, q_2 = 2\ q_1\}$

Several important conclusions follow from this analysis. First, we can generally confirm an improvement in classification accuracy due to value clustering. For example, on average over possible training samples of size 10-20 we can expect to get an improvement of up to 3-4% for certain data models. Note that this is an improvement due to merging of a *single* pair of states and in one-dimensional feature space. Potentially, one can gain more from merging multiple pairs of states.

Second, the actual expected improvement decreases with large sample sizes. This confirms our intuition that the positive effect of value clustering is mainly due to improved estimates of the collapsed meta-states of the variables.

Third, most accuracy gains are achieved when we merge the X variable's states that contribute relatively little to discrimination between classes. Hence, the main challenge of the developed genetic algorithm is to implicitly identify such values and perform their merging into meta-values.

In the next section we consider the empirical results from the application domains that we discussed earlier.

3.6 Experimental Study

The main goal of the empirical study is to compare the performance of select classifiers with and without GA value clustering. Even though an overall accuracy improvement is very valuable, we are also interested in reduction of measurement complexity.

3.6.1 Genetic Algorithm Details

The driver GA program included the following major steps:

1. Initialization. The population is initialized with random individuals using the equalized compact graph-based encoding as described previously in section 3.3. Each individual is created by selecting its genes from the alphabet of respective chromosome positions. For the i-th position, the probability to select the value i is set to $(N\text{-}1)/N$, to prevent formation of a single big cluster covering all the objects. If the value i is not generated, we initialize this gene to one of the remaining available values (edges) for this position using a uniform probability. Prevention of excessive clustering is motivated by a result from graph theory on the critical probability of connectedness of random graphs [13]. Finally, we always seeded a chromosome [1, 2,..., N], corresponding to N non-merged objects (feature values).
2. Fitness evaluation of each individual in the population. Chromosomes are easily decoded to the actual cluster partitioning of feature values as described above. Model parameters are estimated according to the extracted clustering. Classifier accuracy is evaluated using hold-out samples or 10-fold cross-validation.
3. Termination criteria check. The number of generations was the measure of computational effort. The search was stopped after about 200 generations in each run
4. Tournament selection of parents (tournament size = 2). Pairs of individuals are selected at random with replacement and their number is equal to the population size. The better of the two individuals becomes a parent at the next step.
5. Crossover and reproduction. Standard 2-point crossover is used. Each pair of parents produces two offspring. Mutation with small probability $p_m = 0.01$ is applied to each

position. In addition, elitism was always used and the best individual of the population survives unchanged.

6. Continue to step 2.

3.6.2 Experimental Results

Text categorization. The full dictionary of the Reuters data set contains tens of thousands of words (values to be clustered). It is commonly acknowledged that preprocessing is necessary to keep reduce computational costs to a reasonable level. We performed preprocessing by retaining features having the highest mutual information within the category label. Mutual information was assessed independently for each feature. In two different setups we selected 10 and 50 words respectively, corresponding to chromosome lengths with 9 and 49 genes in the equalized compact encoding.

The first set of experiments was designed to distinguish between Reuters articles on "acquisitions" or "earnings" with 2308 and 3785 non-overlapping documents in each category respectively. The known expected accuracy is extremely high, so we made the task more challenging by considering only the first two lines in the body of every document. 1500 documents from each class were split between training and holdout sets (for classifier fitness evaluation); the rest of documents were used to estimate the true accuracy of the best found solution. After the search phase, the best classifier was induced from a combination of training and holdout sets before the final testing. The comparison between the three methods was made, namely, a naïve non-clustered approach, our proposed clustering GA and a pure best-first hill climbing with the same number of fitness evaluations. We performed 100 runs of each algorithm with different

sizes of splits between training and holdout sets. Our GA approach ran for 200 generations with a population of size 100. The results for the estimated true accuracy and percentage of runs with improvement are reported in Table 3.1. Last column shows percentage of runs when GA found better than naive solution. Each row is the average over 100 runs, with training and holdout sets sampled at random in each run from original data.

Table 3.1. Comparison of test accuracy for the best solutions found by GA value clustering and best-first hillclimbing with "naive" non-clustered solution on "*acquisitions*" or "*earnings*" classification problem. First two columns, are training and holdout set sizes.

	Tr. set	Holdout	"Naïve" model	Hill-climbing	Clustering GA	Best runs, %
10 words	400	1100	79.9	79.9	82.3	87
	600	900	79.8	80	81.6	81
	800	700	79.7	79.9	81.5	78
	1000	500	79.9	79.9	81.1	74
50 words	600	900	88.2	88.7	90.8	73
	800	700	88.4	88.8	90.5	60

The results show that GA typically improves the predictive accuracy by 1-2% and most runs are successful. Perhaps the main advantage of the GA was in dimensionality reduction. The average number of clusters for the 10-words problem was 7.9 and 38.6 for the 50-words problem.

Experiments with similar setup were performed to discriminate between the documents in the *gold* and *silver* news categories (topics) on one side and *gas* and *nat-*

gas on the other side. To ensure that our baseline sample contains enough documents we sampled three line subsets from large documents and considered them as individual texts. The classification accuracy averaged over 100 independent runs of the GA is compared to the results of hill-climbing search as well as simple Bayes classifier and reported in Table 3.2.

Table 3.2. Comparison of test accuracy for the best solutions found by GA value clustering and best-first hillclimbing with "naive" non-clustered solution on "*silver*" and "*gold*" versus "*gas*" and "*nat-gas*" topics classification problem.

	Tr. set size	Holdout size	"Naïve" model	Hill-climbing	Clustering GA
10 words	200	200	84.0	85.8	86.5
	250	200	84.5	85.7	86.6
50 words	200	200	89.5	90.0	91.4
	250	200	90.0	90.3	91.3

DNA fingerprinting. The experiments were done with multiple data sets. Each data set contained individuals drawn from two populations. The data was collected from a simulation program that can generate individuals from the source populations with a user-chosen level of inter-population dispersal as measured by the Wright's F-statistic values, in particular, the value of F_{st}. We used 100 individuals per sampled population as training data. Each individual is given as a set of two alleles per locus (feature) with 8 loci in total. While most of the features had only 2-4 different alleles, there were 10 possible alleles for one of the loci. We ran the GA value clustering only for the allelic values at this particular locus. The classification step and GA search are very similar to what was done for text categorization. However, the probabilities of individuals are computed differently, because for Mendelian genotypes there are two alleles specified at each locus

(feature). Namely, the probability of allelic configuration (i,j) at each locus is estimated as $2p_ip_j$, if alleles i and j are different, or as p_i^2, if $i=j$, where p_i is an estimate of i-th allele probability at a locus. This modification of the regular naïve Bayes classifier is trivial and is necessary only in computations of fitness for candidate solutions. Also the fitness was estimated on completely independent test set of 100 individuals. As a main result, the value clustering GA was able to improve classification accuracy (cross-validated) from 69% to 73% for low Fst data (considerably overlapping populations), from 91% to 92% for high Fst data (populations with little overlap). In most runs, the best solution was consistently represented by only 5-7 meta-values instead of the 10 original values (alleles).

Bayesian Network Optimization by Value Clustering. The goal of value clustering as applied to Bayesian network is to intelligently modify conditional probability tables in order to improve accuracy of classification. In our experiments we followed these steps.

1. Learning the structure of Bayesian Network Classifier: the main purpose of structural learning is to provide a tree of dependencies over attribute nodes. Class-conditional mutual information (CMI) between attributes serves as major guiding principle for better structure. Only links with highest value of CMI are kept. It means that a maximal spanning tree is constructed from the complete graph, where edges are weighted by CMI computed from training data. It was shown in [18] that such a tree is optimal among all other possible trees over attribute nodes. The proof shows that maximum spanning tree maximizes the likelihood function for given data set.

2. Parametric training from the given training data. This includes estimation of all required conditional probability tables (CPTs) for the nodes present in the given fixed structure found. Every CPT is either three-dimensional $P(x|y,c)$ or two-dimensional $P(x|c)$. If the only incoming link is from the parent class node then we have the usual parameter estimation as in case for naïve Bayes; i.e., we estimate assuming a multinomial distribution for the values of x. Learning the values of the CPT table is straighforward from the data provided, because it is done in linear time with respect to sample size. During parametric training we compute the observed frequencies of values of the required attribute conditioned on the combinations of values of other attributes as well as class label. This gives us the maximum likelihood estimate of the parameters (CPT entries) for a fixed network structure.

3. Genetic value clustering. We use a genetic algorithm described earlier to optimize the performance of a BNC. The GA is applied to CPT tables at the nodes of the network. Clustering of both rows and columns is done for one node at a time. We perform value clustering for two nodes at most, sequentially.

4. Testing of trained BNC. At this phase we compute posterior probabilities for each class according to the learned structure and CPTs. The maximum value of posterior probability decides class membership for every test instance.

5. Editing of BNC can be done after training. Links can be deleted from the maximum spanning tree. Essentially it means that we can consider an arbitrary forest over attribute nodes. Usually we had one or two edges remaining after editing.

6. Smoothing control is achieved via combination values of weights for $P(x|c)$ and $P(x|y,c)$. The idea is to mix the actual value of conditional probability with its prior value.

Though this method was first introduced in Bayesian statistics in the late eighties, we rediscovered it for our application independently. The idea is to balance between full link and absence of link in a TAN structure.

$$\tilde{P}(x \mid y,c) = \alpha P(x \mid c) + (1-\alpha)P(x \mid y,c),$$

where y is a parent of x and α is weight. If $\alpha = 1$, this is equivalent to a naïve Bayesian classifier. If $\alpha = 0$, it gives us a general TAN. In our experiments with smoothing below, the value $\alpha = ½$ is used to gain benefits from both.

Testing of the Bayesian Network classification algorithms with value clustering was done using two types of datasets: 4 datasets from the Machine Learning UC Irvine repository and 3 datasets generated by the population genotype simulator written by the author. All data was discrete and had 3-6 features with 3-4 possible states per feature. Instances were drawn from 2 or 3 classes. Table 3.3 provides necessary details about the datasets.

Table 3.3. Characteristics of the datasets used in BNC experiments

Datasets	Number of samples in training set	Number of samples in test set	Number of attributes	Number of classes
Hayes-Roth	132	28	3	3
Monk-1	124	432	6	2
Monk-2	169	432	6	2
Monk-3	122	432	6	2
Moderate	200	200	6	2
Strong	200	200	6	2
Weak	200	200	6	2

Monk's test problems are often used in machine learning research. The instances of this dataset describe robots by the following nominal features:

Head shape $\in$ {round, square, octagon}, Body shape $\in$ {round, square, octagon}

Is smiling? $\in$ {yes, no}, Holding $\in$ {sword, balloon, flag}

Jacket colour $\in$ {red, yellow, green, blue}, Has tie? $\in$ {yes, no}

Monk1 target hypothesis: (head shape = body shape) or (jacket colour = red).

Monk2 target hypothesis: Two and only two of the features assume their first listed value.

Monk3 target hypothesis: (jacket-colour = green and holding = sword) or (jacket-colour $\neq$ blue and body-shape $\neq$ octagon).

Another problem taken from Irvine repository is the Hayes-Roth dataset. The target concept behind this dataset were described as following:

Class 1: (# of 1's)>(# of 2's) for attributes 1-3

Class 2: (# of 2's)>(# of 1's) for attributes 1-3

Class 3: if 4 occurs for any attribute

Apparently, decision trees are appropriate in recovering such generation rules due to the similarity of the hypothesis spaces. That is why we additionally ran all the tests through the decision tree program C5. Three genetic data sets were created by the custom program that sets the desired feature distributions (allelic frequencies) as well as degree of correlation among the features. We created data sets with weak, moderate and strong linkage. The features were pairwise correlated, namely each attribute was linked at most to one more attribute. For example, the relations were 1-2, 3-4 and 5-6. Other, possible pairs of features were considered independent. Therefore, we had good prior knowledge of the distributions and were able to compare learned BN classifier structure with our assumptions. Expected structure of classifiers for custom data sets is shown on Figure 3.15.

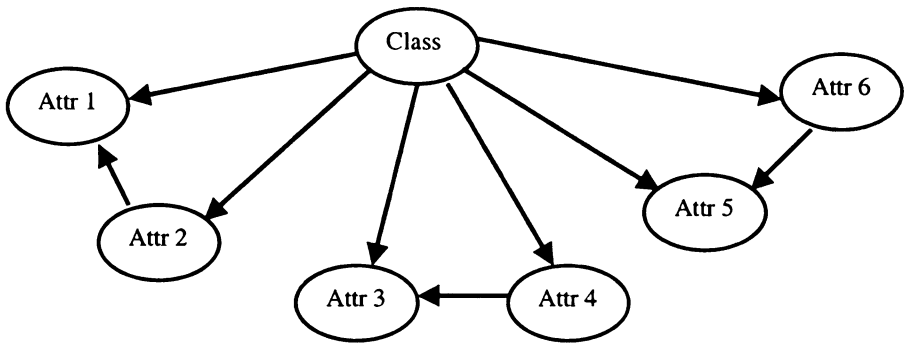


Figure 3.15. Underlying dependencies between the variables in the datasets "weak", "moderate" and "strong".

Table 3.4. Classification accuracy of the compared classifiers over test sets (no smoothing)

Data Set	*DT C5.0*	*Naïve Bayes*	*BNC-TAN 1 with GA*	*BNC-TAN 2 with GA*
hayes-roth	92.9	88. 8	91.4	88.9
Monk-1	74.3	71.3	73.1	72.7
Monk-2	65.0	54.7	59.6	53.2
Monk-3	97.2	97.2	97.2	93.5
Strong	77.0	76	77.0	73.5
Moderate	77.0	72.5	76.5	71.0
Weak	66.0	70	70.0	67.5

Table 3.5. Classification accuracy on test sets with classifier smoothing (α=½).

Data Set	*Naïve Bayes*	*BNC-TAN 1 with GA*
hayes-roth	88.8	90.3
monks-1	71.3	73.4
monks-2	54.7	59.1
monks-3	97.2	97.2
strong	76	76.5
moderate	72.5	73.0
weak	70	70.0

The experimental results indicate that value clustering has been able to improve on the results of naïve Bayes classifier for almost all datasets. Though the decision trees seem to outperform probabilistic models on this data, this is to be expected since their decision rules match the target concepts very well. The genetic value clustering performed nearly as well with probabilistically generated data with limited attribute dependencies.

3.7 Conclusions

The novelty of the approach is in adopting a genetic algorithm for clustering the values of variables. In contrast, traditional genetic algorithm approaches for data mining operate on the features as a whole by including/excluding them from a subset or adjusting the appropriate weights. We take a further step and organize the search in the entire input space. The enhanced graph-based encoding has a much less redundant chromosome while maintaining complete coverage of possible partitions, as we strictly prove. The experiments performed demonstrate reduction in measurement complexity and improvement in classification accuracy due to better reliability of meta-values.

It is possible to generalize and apply genetic clustering to a difficult problem of parametric learning in the context of Bayesian networks. Bayesian network classifiers, even augmented to trees over the feature nodes, require estimation of a much greater number of parameters than a typical naïve Bayes classifier. We demonstrate that the conditional probability distribution in a Bayesian network can be considerably simplified and learned from data with the help of GA value clustering.

Chapter 4

Genetic Programming with Local Learning

Data mining deals not only with discrete variables, but also with continuous values of input and output variables. In this case a solution to a classification problem can be obtained in the form of a model that relates continuous features in input and output. Building a model, in fact, amounts to learning a function fitting given data samples, which is an interpolation problem. Genetic programming is a modern tool for learning arbitrary models from the given data using a special set of building blocks – node functions.

The quest for more efficient Genetic Programming is an important research problem. This is due to the fact that the high computational complexity of GP is among its distinctive features [104]. Especially now, when variants of GP are being used on very ambitious projects [78][127], the speed and efficiency of evolution are very crucial for such problems.

Numerous modifications of the basic GP paradigm [75] are currently known, e.g. see [80] for a review. Among them, several researchers have considered GP augmentation by hill climbing, simulated annealing and other stochastic techniques. In

[100] crossover and mutation are used as move operators of hill climbing, while Esparcia-Alcazar and Sharman [30] considered optimization of extra parameters (node gains) using simulated annealing. Terminal search was employed in [140], but due to the associated computational expense it was limited to 2-4% of individuals. The presence of stochasticity in local learning makes it relatively slow, even though some hybrid algorithms yield overall improvement. Iba and Nikolaev [62] and Rodriguez-Vazquez [115] considered least squares coefficient fitting limited to linear models. Apparently, the full potential of local search optimization is yet to be realized.

The focus of this study is on a local adaptation of individual programs during the GP process. We rely on gradient descent for improved generation of GP individuals. This adaptation can be performed repeatedly during the lifetime of an individual. The results of local learning may or may not be coded back into the genotype (reverse transcription) based on the modified behavior, which is reported in the literature as Lamarckian and Baldwinian learning, respectively. The resulting new fitness values affect the selection process in both cases, which in turn changes the global optimization performance of a GP. Such an interaction between local learning, evolution and associated phenomena without reverse transcription is also generally referred to as the Baldwin effect.

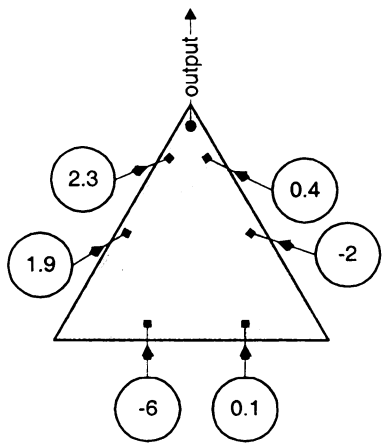


Figure 4.1. Sample tree with a set of random constants. In hybrid GP all the leaf coefficients are subjected to training.

We were motivated by a number of successful applications of hybridization to neural networks [8][147][97]. Both neural networks and GP trees perform input-output mapping using a number of adjustable parameters. In this respect, terminal values (leaf coefficients) in a GP tree serve a similar function to weights in a neural network. A form of gradient descent is usually used to adjust weights in a neural net architecture. In contrast, various terminal constants are typically random within GP trees and are rarely adjusted by gradient methods. The reasons for this are twofold: the unavailability of gradients/derivatives in some GP problems and the computational expense that is assumed to exist in computing those gradients. However, the complexity of computing derivatives is largely overestimated. In order to differentiate programs explicitly, algorithmic differentiation [51] may be adopted. Algorithmic (computational) differentiation is a technique that accurately determines values of derivatives with essentially the same time complexity as found in the execution of the evaluation function itself. In fact, gradients may often be computed as part of the function evaluation. This is

especially true for trees and at least potentially true for arbitrary non-tree programs. Generalization of the algorithmic differentiation method for any program is possible, given that the generated program computes numeric values, even in the presence of loops, branches and intermediate variables. The main requirement is that the function be piecewise differentiable. While not always correct, this is the case for a great majority of engineering design applications. Moreover, it is also known that directional derivatives can be computed with many non-smooth functions [51]. Knowledge of only gradient direction, not its value, is often enough to optimize the values of parameters.

In this work we empirically compare conventional GP with a GP coupled with terminal constant learning. The effectiveness of the approach is demonstrated on several symbolic regression problems. Arithmetic operations have been chosen as the primitives set in our GP implementation for simplicity's sake. While such functions make differentiation easy, again, these techniques can be adapted to more difficult problems.

Our results indicate that inexpensive differentiation, along with Baldwin effect learning, leads to a very fast form of GP. Significant improvement in accuracy is achieved beyond that which could be achieved by either local search or more generations of GP.

The Baldwin effect is known to change the inductive bias of the algorithm [134]. In the case of GP, where functional complexity is highly variable, it is expected that such a change of bias can be properly quantified. Two manifestations of the learning bias were observed in our experiments. Firstly, the selection process is affected by local learning since the fitness of many individuals dramatically improves during their lifetime. Secondly, changes in the functional complexity of individuals were observed in the

experiments. Both the length (number of nodes) of the best evolved programs and the number of leaf coefficients were higher using local learning as opposed to regular GP.

4.1 Lamarckian versus Baldwinian Strategy in GP

Evolution rarely proceeds without phenotypic changes. As we are interested in digital evolution, two dominating strategies have been proposed which allow environmental fitness to affect genetic features. Lamarckian evolution, an alternative proposition to Darwinian approaches of the time, claimed that traits acquired from individual experience could be directly encoded into the genotype and inherited by offspring. In contrast, Baldwin claimed that Lamarckian effects could be observed where no direct transfer of phenotypic characteristic to the genotype occurred, in keeping with Darwinism. Rather, Baldwin claimed that "innate" behaviors could be selected for (in a Darwinian sense) which the individual originally had to learn. In Lamarckian evolution learning affects fitness distribution as well as the underlying genotypic values, while the Baldwin effect is mediated via the fitness results only. In our case, the question is whether locally learned constants are copied back into the genotype (Lamarckian) or whether the constants are unmodified while the individual's fitness value reflects the fitness resulting from learning (Baldwin).

Real algorithmic implementations of evolution coupled with local learning are much richer than the two original strategies. The researcher, usually guided by the total computational expense, may arbitrarily decide both the amount of and scheduling of learning or local adaptation of solutions. Moreover, since local learning comes with a

price, it must be wisely traded off with genetic search costs. Several questions must be answered:

- What aspect of the solution should be learned beyond genetic search, as only a subset of solution parameters may be chosen for adaptation?
- Should learning be performed at every generation or should it be used as a form of fine-tuning when genetic search is converged?
- How many individuals and which of those individuals should have local learning applied to them?
- How many iterations of local learning should be done (really, how much computational cost are we willing to incur)?

Accordingly, there are many ways to introduce local learning into GP. Evolution in GP is both parametric and structural in nature. Two important features are specific to GP:

1. The fitness of the functional structure depends critically on the values of local parameters. Even very fit structures may perform poorly due to inappropriate numeric coefficients.
2. The fitness of the individual is highly context sensitive. Slight changes in structure dramatically influence fitness and may require completely new parameters.

That is why we focus on learning numeric coefficients, so called Ephemeral Random Constants or ERC [75], which are traditionally randomly generated as shown in Figure 4.1. As explained below, the local learning algorithm – gradient descent on the error surface in the space of the individual's coefficients, turns out to be a very inexpensive approach, so much so that every individual can do local learning in every generation.

Formally we follow the Lamarckian principle of evolution since we allow the tuned performance of individual to directly affect the genome by modifying numeric constants. At the same time, the choice between Lamarckian and Baldwin strategies in our implementation is not founded on the issue of computational complexity. In both cases the amount of the extra work is approximately the same. The main issue arises when considering the fitness values of the offspring with inherited coefficients vs. offspring with unadjusted terminals. Our experiments indicate that there is little difference between these two fitness values when crossover is the main operator. Two factors contribute to this phenomenon:

1. Crossover usually generates individuals with significantly worse fitness than their parents. The coefficients found earlier to be good for the parents are not appropriate for the offspring structures. The subsequent local learning changes fitness dramatically by updating the ERCs to more appropriate values.
2. Newly generated offspring are equally well adjusted starting from any values: earlier trained, not trained or even random.

Hence, inheritance of the coefficients does not much help the performance of the individuals created by crossover. However, if an individual is transferred to a new generation as a part of the elitist pool, i.e. unchanged by crossover or mutation, then its learned coefficients are also transferred. With respect to this structure, the use of the Baldwinian strategy would be wasteful, since it requires relearning the same parameters. Thus, even though our implementation formally follows the Lamarckian strategy, we effectively observe the very same phenomena peculiar to the Baldwin effect.

4.2 Hybrid Genetic Programming

The organization of the hybrid GP (HGP) in many respects is the same as that of the standard GP. The only extra activity done by the HGP algorithm is to update the values of numeric coefficients. That is, all individuals in the population are trained using a simple gradient algorithm in every generation of the standard GP. Below we discuss the exact formulation of the corresponding optimization problem.

The hybrid GP is intended to solve problems of a numeric nature, which may include regression, recognition, system identification or control. We will assume throughout that there are no non-differentiable nodes, such as Boolean functions. In general, given a set of N input-output pairs $(d,\mathbf{x})_i$ it is required to find a mapping $f(\mathbf{x},\mathbf{c})$ minimizing certain performance criteria, e.g. mean squared error (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(d_i - f(\mathbf{x}_i,\mathbf{c})\right)^2$$

Here, f is scalar function (generalization to multi-trees is trivial), $\mathbf{x}$ is vector of input values, $\mathbf{c}$ is vector of coefficients and sum goes over the training samples. Of course, in GP we are interested in discovering the mapping $f(\mathbf{x}, \mathbf{c})$ in the form of a program tree. That is, we seek not only coefficients $\mathbf{c}$, but also the very structure of the mapping f, which is not known in advance. In our approach, finding the coefficients is done by gradient descent during the same time functional structures are evolved. Descriptions of the standard GP approach can be found elsewhere (e.g. [80]), instead, we will focus below on details of the local learning algorithm.

4.3 Learning leaf coefficients

Minimization of MSE is done by a few iterations of a simple gradient descent. At each generation all numeric coefficients are updated several times using the rule:

$$c_k \rightarrow c_k - \alpha \frac{\partial MSE(\mathbf{c})}{\partial c_k},$$

where α is the learning rate, and k goes over all the coefficients at our disposal. Three important points must be discussed: how to find the derivatives, what the value of α should be, and how many iterations (steps) of descent should be used.

From two previous equations we obtain:

$$\frac{\partial MSE(\mathbf{c})}{\partial c_k} = -\frac{2}{N}\sum_{i=1}^{N}\left(d_i - f(\mathbf{x}_i, \mathbf{c})\right)\frac{\partial f(\mathbf{x}_i, \mathbf{c})}{\partial c_k}$$

Thus, an immediate goal is to differentiate any current program tree with respect to any of its leaves. The chain rule significantly simplifies computing $\partial f/\partial c$. Indeed, if $n_j(\cdot)$ denotes node functions, then:

$$\frac{\partial f(n_1(n_2(n_3(...),...),...))}{\partial c_k} = \frac{\partial f}{\partial n_1}\frac{\partial n_1}{\partial n_2}\frac{\partial n_2}{\partial n_3}\cdots\frac{\partial n_r(c_k,...)}{\partial c_k}$$

Therefore differentiation of the tree simply reduces to the product of the node derivatives on the path which starts at the given leaf and ends at the root. It is clear that each term in the product is a derivative of a node output with respect to its arguments (children). If the paths from the different leaves share some common part, then corresponding sub-chains in the derivatives are also shared. Computation of such a product in practice depends on the data structure used for the program tree. In simple cases, differentiation uses a single recursive postorder traversal together with the actual function evaluation. Derivatives of

the program tree with respect to all its leaves can be obtained simultaneously. As soon as an entire sum in computing leaf coefficient correction becomes known, i.e. derivatives in all training points obtained, one may need an extra sweep through the tree to update the coefficients. In total, the incurred overhead depends on the complexity of node derivatives and the number of leaves. For instance, in our implementation using only an arithmetic functional set, the cost of differentiation was equal to the cost of function evaluation, making the overall cost twice the standard GP cost for the same problem.

4.4 Program Differentiation in Detail

Let us consider details of on-the-fly differentiation of programs represented by trees. Potentially this technique can be generalized to programs of other structure as proved in the area of *algorithmic differentiation*. We will assume that first-order derivatives for the node functions exist. Program tree can be viewed as a complex nested function: $f(n_1(n_2(n_3(...),...),...))$, where $n_i(x,y)$ is a node function from a chosen functional set. For example, the tree shown in Figure 4.2 has equivalent representation as a function:

$$f(x, y, z, c_1, c_2) = ((x \times y) + c_1) \times ((x - z) - (y \times c_2))$$

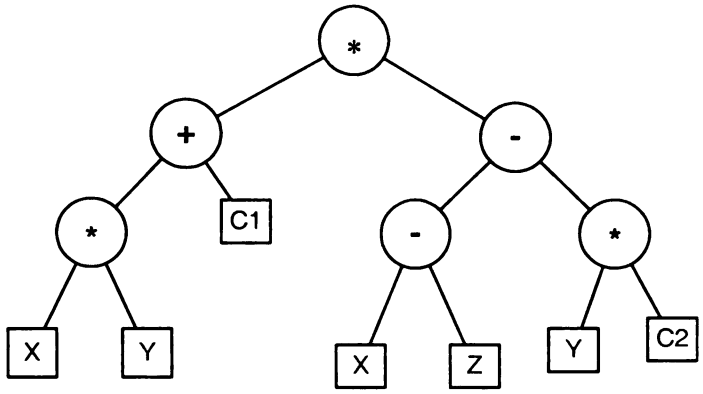


Figure 4.2. Example of a function of 3 variables drawn as a tree. C_1 and C_2 are the leaf coefficients

Here the node functions have two arguments. Nodes with only one argument can be included too. Any node function with more than two arguments is trivial to decompose if necessary. Though for the purpose of differentiation, it is only important to know the derivatives of every function in the set with respect to each of its arguments (no matter what is the complexity of the function).

The overall goal of GP is to find a function, which is the best fit for the training data. Assuming that input is represented as $\boldsymbol{x}$ and desired output as y then it is required to find a mapping $f(\mathbf{x},\mathbf{c})$ minimizing mean squared error:

$$\min_{f,c} MSE = \min_{f,c} \sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i, \mathbf{c})\right)^2$$

The sum goes over all given input-output pairs $\{y_i, \boldsymbol{x}_i\}$, and $\boldsymbol{c}$ is a set of parameters. During GP search we are seeking both the structure and corresponding parameters. The essence of the problem is how to find the best values of parameters $\boldsymbol{c}$ for a function f which is not known in advance but is generated. In fact, we have many such functions, because we work with the population of programs .If we are going to apply a gradient

descent then we have to be able to find the derivatives of this function. At first look, it seems to be difficult, because the structure of f is not fixed, but created dynamically during the search. However, *algorithmic differentiation* states that if one can compute the value of a function, then we can compute a value of its derivative. The key, of course, is that the programs are "computable", i.e. comprised of elementary operations. Therefore this provides an opportunity to find the values of derivatives not approximately, but exactly. This is important, since no estimation error is introduced, as it could be in case of some sort of approximation, e.g. by finite differences. Also, the values of derivatives can be found by essentially using the same code that represents the function. The only error in the values is simply due to the nature of numerical computations.

For example, if we evaluate a function $f(x,y) = x^2 + xy$, at point $x=1$, $y=0$, then we hope to obtain the value of the derivate with respect to x: $f_x'(1,0)=2$, or with respect to y: $f_y'(1,0)=3$, in addition to the value of the function itself: $f(0,0)=1$.

In general, in order to decrease the mean-squared error we have to adjust the value of function's parameter c_k, by making a step in the direction opposite to gradient:
Such adjustments can be made for all or only some of the available parameters $\{c_k\}$. As we already know, in order to compute an adjustment to the parameter c_k it is required to obtain $\frac{\partial f(\mathbf{x}_i,\mathbf{c})}{\partial c_k}$ and $f(\mathbf{x}_i,\mathbf{c})$. Note that the output of a function $f(\mathbf{x}_i,\mathbf{c})$ is computed for fitness evaluation anyway. Hence, it is desired to accomplish computation of $\frac{\partial f(\mathbf{x}_i,\mathbf{c})}{\partial c_k}$ at the same time.

Let us consider the simultaneous computation of function and its derivative with respect to one parameter c. The free parameter c, is shown explicitly in Figure 4.3, while the rest of the function is assumed to be sitting within the triangles and therefore is not relevant at the moment. Differentiation by c gives:

$$\frac{\partial f}{\partial c} = \frac{\partial n_1(n_2(n_3(n_4(c,...),...),...),...)}{\partial c} = \frac{\partial n_1}{\partial n_2}\frac{\partial n_2}{\partial n_3}\frac{\partial n_3}{\partial n_4}\frac{\partial n_4(c,...)}{\partial c}$$

This example shows that to compute a derivative of a function, one must multiply partial derivatives on the path from c to the root. Let us work out the example in more details. Consider the function $f(x,y,c)= x*(c(x+y)-xy)+y$. Suppose we would like to compute the

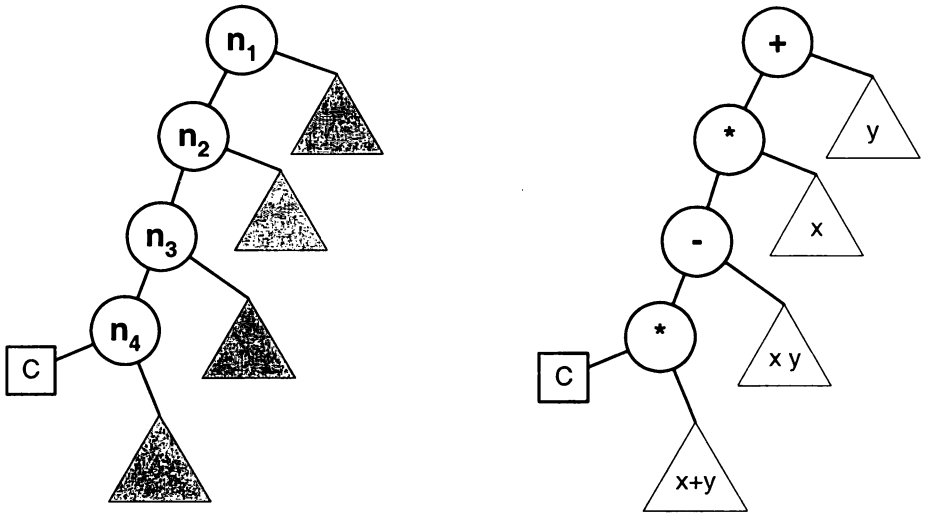


Figure 4.3. Example of a function for algorithmic differentiation with respect to a parameter c.

values f(x=1,y=2,c=1) and $\left.\frac{\partial f(x,y,c)}{\partial c}\right|_{(1,2,1)}$. Again, we assume that all the triangles are arbitrary subtrees not depending on parameter c.

Without the knowledge of analytical expression of a function f we compute it by traversing the tree: if we store this function also as a tree, then we can evaluate it by postorder traversal. However, it is required to introduce the auxiliary variable '*df*',

associated with the parameter c, in which we will keep the value of the derivative. Hence we obtain:

1. f=0, df=0, n_4_left =1, n_4_right=3
2. f=3 (=n_4(n_4_left=1, n_4_right=3)), df=3, because we know that derivative of this '*' node with respect to the left argument equals to the right argument. In general, we make the distinction between the "left" and "right" derivatives, e.g. for "-" (minus) or other asymmetric node functions.
3. f=1 (=n_3(n_3_left =3, n_3_right=2)), df=3 (=3*1=df*1), because the "left" derivative of "-" node is 1.
4. f=1 (=n_2(n_2_left =1, n_2_right=1)), df=3 (that is =3*1=df*1= df* n_2_right)
5. f=3 (=n_1(n_1_left =1, n_1_right=2)), df=3 (that is =3*1=df*1)

Therefore the result computed is: $f(x=1, y=2, c=1) = 3$ and $\left.\frac{\partial f(x,y,c)}{\partial c}\right|_{(1,2,1)} = 3$. If there is more then one point in the training set, then we just found one term from the sum over the dataset and need to repeat the same computations at other points (x_i, y_i), i=1..N. In this situation we do not adjust the parameter c yet, but instead accumulate the value of correction in some intermediate variable, say 'delta'. Once, all terms in the sum over the training set are found, then we finally update the parameter c.

Now it is clear how the general scheme works: we compute the value of the derivative at the same time as we compute the function and using the same program flow as the function itself.

There are many implementation dependent issues, which may influence how it is done technically. E.g. how we allocated the 'delta' variable: if 'delta' is a pair (structure) of a value and a pointer to c, then in order to update c, we don't need to traverse tree again to

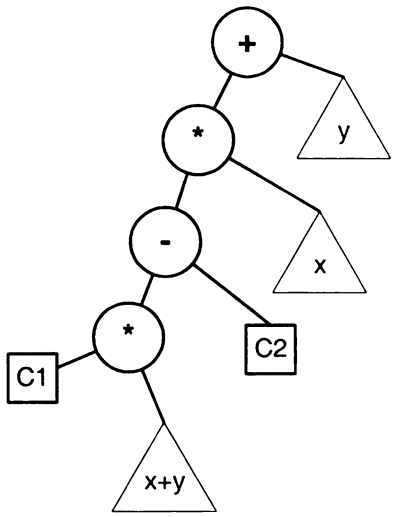


Figure 4.4. Example of a function for algorithmic differentiation with respect to 2 parameters c_1 and c_2.

find c. Also, the nodes now become pointers to structures, which keep pointers to not only the node functions, but also to its derivatives (e.g. to "left" and "right" derivatives for two-argument functions). Of course, we have to supply the GP with functional set and corresponding set of derivatives.

Now let us consider a more complex example, where we have more than one parameter. If we want to find derivatives with respect to several parameters than we also can do it on the same pass through the tree. Suppose there are two parameters c_1 and c_2. In this case we have to carry three variables while traversing through the tree, f, df_1, df_2. We initialize the value of f at the very beginning of evaluation, and allocate df_i only if necessary.

There are some other potential savings because of the tree-specific structure. The paths from c_1 and from c_2 share common subpath to the root node. This means that the values of some terms in the product of the chain derivative can be re-used for both derivatives. Note that two terms $\frac{\partial n_1}{\partial n_2}\frac{\partial n_2}{\partial n_3}$ are common in both derivatives. In fact, when

df_1 and df_2 are computed, their values will be multiplied by the same factors as soon as their paths meet. For example, here the paths from c_1 and c_2 meet after the node n_3. The scheme can be generalized for any number of parameters – all the derivatives can be computed on the same pass through the tree. It should work not only with trees but with almost any program, because computations of derivatives mirror program flow. If, for example, we have conditional control like "IF" or "FOR" then we only need to carefully keep track of derivatives. In this case the implementation can be more complex, because with "FOR" loops we could repeatedly go through the same "terminals" containing the parameters, and therefore should avoid re-initializing the auxiliary variables like *df*.

In order to adjust the parameters ***c***, sometimes we can sacrifice precision, because we do not rely very much on the actual value of derivatives, but rather on their sign. That is, we mostly need to make a step in right direction, to either increase or decrease the value of a parameter. Therefore, even if exact derivatives are not known (or don't exist), but only "directional" derivatives are known, we still can use them.

4.5 Learning rate and number of steps

In a simple gradient descent algorithm, the proper choice of learning rate is very important. Too large a learning rate may increase error, while too small a rate may require many training iterations. It is also known that the simple gradient Cauchy rule works better in the areas far from the vicinity of local minima [114]. Therefore we decided to make the rate as large as possible without sacrificing the quality of learning. After a few trials on the test problem of symbolic regression we fixed the learning rate to

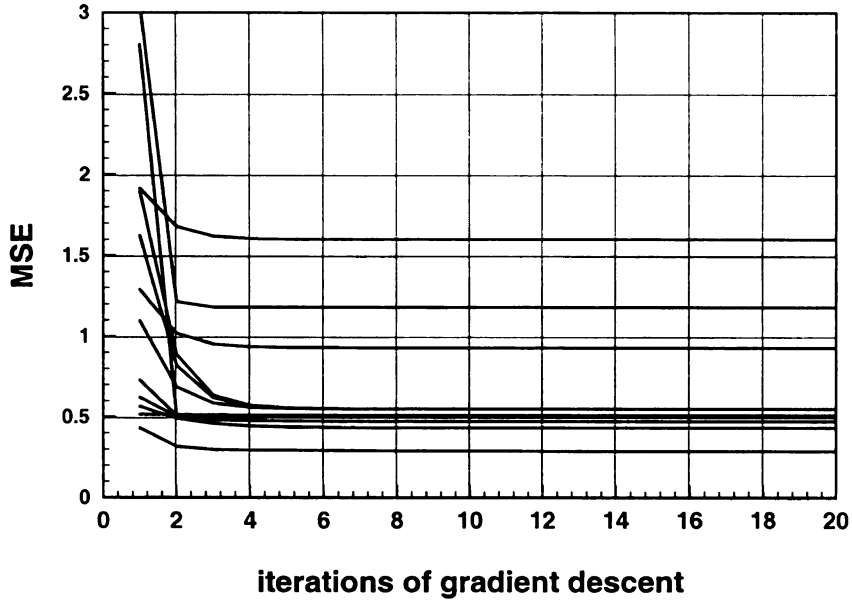


Figure 4.5. Local learning strongly affects fitness of individuals. Typical learning progress is illustrated using individuals from test problem f_2.

the value α=0.5. The same learning rate was used for all other test problems. If the algorithm resulted in an increase in the error of an individual, the training was stopped and no update to the individual's fitness was recorded. However, this problem did not have any impact on overall quality of learning since it happened rarely, approximately 1 out of 10 successful individuals. Moreover, those individuals that had this problem showed an error rate that was typically not reduced by any subsequent application of gradient descent.

This simple local learning rule dramatically improved the fitness of individuals. Figure 4.5 shows the decrease in MSE for typical individuals. It is important to note that the most significant improvements happened after only the first few iterations of local learning. Note that some individuals were improved by as much as 60% or more. We decided that 3 steps of gradient descent was a good trade off between fitness gain and

effort overhead. Again, the number of iterations was never altered afterwards and is used in all our experiments.

4.6 Experimental Design

The main goal of the empirical study is to compare the performance of the GP with and without gradient-descent learning. Even though an overall speed-up is very valuable, we are also interested in other effects resulting from such learning. These effects have to be properly quantified to shed light on the internal mechanisms of the interaction between learning and evolution. Three major issues are studied:

- Improvement in search speed
- Changes in fitness distribution and selection
- Changes in the functional structure of the programs

The driver GP program included following major steps:

1. Initialization of the population using the "grow" method. Starting from a set of random roots, more nodes and terminals are aggregated with equal probability until a specified number of nodes are generated. The total number of nodes in the initial population was chosen to be three times greater than the population size, which is three functional nodes per individual on average.
2. Fitness evaluation and training (in HGP) of each individual. Mean squared error over the given training set, as defined before, serves as an inverse fitness function since we seek to minimize error. This stage includes parametric training in HGP given that the individual has leaf coefficients.

3. Termination criteria check. The number of function evaluations was the measure of computational effort. For instance, every individual is evaluated only once in every GP generation, but three times in every HGP generation if its parameters are trained for three steps.
4. Tournament selection (tournament size = 2) of parents. Pairs are selected at random with replacement and their number is equal to the population size. The better of the two individuals becomes a parent at the next step.
5. Crossover and reproduction. Standard tree crossover is used. Each pair of parents produces two offspring. Mutation with small probability p_m=0.01 is applied to each node. In addition, elitism was always used and the best 10% of the population survive unchanged.
6. Pruning the trees with the size exceeding predefined threshold value (24 nodes for test problems described below).
7. Continue to step 2.

Test Problems

Five surface fitting problems were chosen as benchmarks.

$$f_1(x, y) = xy + \sin((x-1)(y+1))$$

$$f_2(x, y) = x^4 - x^3 + y^2/2 - y$$

$$f_3(x, y) = 6\sin(x)\cos(y)$$

$$f_4(x, y) = 8(2 + x^2 + y^2)^{-1}$$

$$f_5(x, y) = x^3/5 + y^3/2 - y - x$$

For each problem 20 random training points (fitness cases) were generated in the range [-3...3] along each axis. Figure 4.8 - 4.12 show the desired surfaces to be evolved.

4.7 Experimental results

To compare the performance we made experiments with both hybrid and regular GP with the same effort of 30,000 function evaluations in each run. All experiments were done with a population size of 100 and the arithmetic operators {+,-,*,%-protected} as the function set with no ADFs. Initial leaf coefficients were randomly generated in the range [-1...1]. Also, the pruning threshold was set to 24 nodes. If the number of nodes in an individual grew beyond this threshold, a sub-tree beginning at some randomly chosen node was cut from the individual. Each experiment was run 10 times and the MSE value was monitored.

Our main results are shown in Figures 4.8-4.9 and also summarized in Table 4.1. The success of the hybrid GP is quite remarkable. For all the test problems, the average error of the best evolved programs was significantly smaller (1.5 to 25 times) when learning was employed. The first 20 – 30 generations usually brought most of these improvements. The gap in error levels is wide enough to require the regular GP to use hundreds more generations to achieve similar results. Certain improvements were also observed for the average population fitness, but with lesser magnitude. The similarity of each population's average fitness indicates a high diversity and that not all offspring reach small error values after local learning.

Another set of experiments included extra fine-tuning iterations performed only after the regular GP terminates. Again, we run gradient optimization on the population

from the last GP generation. Each individual was tuned by applying 100 gradient descent iterations. The results in Table 4.1 show that this approach is not effective and did not achieve the quality of result found in the HGP. This is a strong argument for Baldwin effect, namely that another factor affecting search speed-up is a change in fitness distribution that directly affects selection outcome. Learning introduces a bias that favors individuals that are more able to adapt to local learning modifications. If we would suppose that the selection bias does not occur, then the hybrid GP would be only a trivial combination of genetic search and fine tuning. However, as we see from the results this is not the case.

We attempted to measure some properties of HGP that would demonstrate this synergy between local learning and evolution.

Table 4.1. Performance Comparison of Hybrid and Regular GP. All data collected after 30000 f.e. and averaged over 10 experiments.

Test problem	Best MSE		Ave. MSE		Best MSE
	HGP	GP	HGP	GP	GP + fine tuning
f_1	0.009	0.26	0.47	0.80	0.233
f_2	0.075	0.761	1.03	2.18	0.31
f_3	2.32	6.22	5.98	6.59	6.21
f_4	0.64	0.76	4.06	4.41	0.76
f_5	0.097	0.36	0.27	0.78	0.30

First of all, a Baldwin effect in selection would mean that the results of some tournament selections are reversed after local learning. Indeed, local learning adaptable individuals

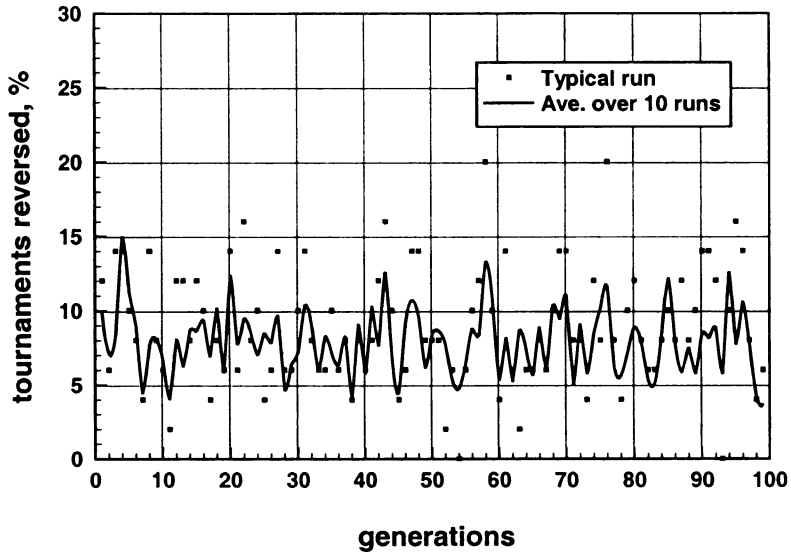


Figure 4.6. Comparison of Selection Process in HGP and GP. Local learning changes outcome of some tournaments used to select a mating pool.

win their tournaments due to improved fitness resulting from gradient descent. These individuals would lose the same tournament in regular GP. Figure 4.6 shows both the typical and average percentage of reversed tournaments in the problem f_1. A summary of results for all the test problems is given Table 4.2.

It was found that the average percentage of selection changes remains the same during the course of search for all test problems. Such a behavior would be expected if selection pressure pushes offspring that are very adaptable, even when older elite members are almost converged. An empirical measure of this degree of adaptability is provided by the average gain in fitness achieved by newly generated offspring. The values are given in Table 4.2. We do not include elite members in this statistic to emphasize magnitude of learning from scratch. The average observed drop of MSE is between 12% and 19% on all the test problems.

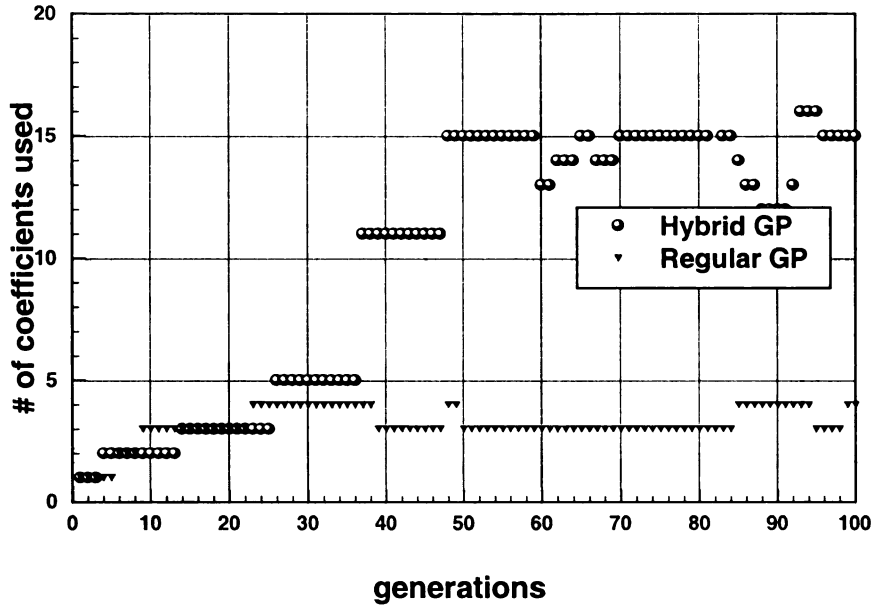


Figure 4.7: Typical dynamic of number of terminals (numeric coefficients) used by the best program as a function of GP generations (for the test function f_1).

What exactly makes one program more adaptable than the other? Clearly, it is the functional structure of the program. For example, a program with no numeric leaves cannot learn at all using the gradient local learning method described above. Furthermore, a tree with no terminal arguments (inputs) containing only terminal constants will always produce the same output and will not benefit from local learning. Instead we have tried to understand what characteristics of adaptable programs are unique.

Table 4.2. Effects of local learning

Test problems	Difference in HGP selection vs. GP in each generation on average, %	Ave. MSE gain for newly generated offspring, %	Complexity of the best programs, #coefficients / #nodes after the same effort (30000 f.e.)	
			HGP	GP
f_1	7.7	16.5	16.0 / 22.4	12.2 / 21.2
f_2	7.1	12.7	16.6 / 23.0	13.7 / 21.8
f_3	8.4	15.1	17.5 / 23.5	11.8 / 20.4
f_4	7.9	18.7	17.3 / 22.9	12.4 / 21.6
f_5	7.4	15.0	17.0 / 23.1	12.9 / 21.8

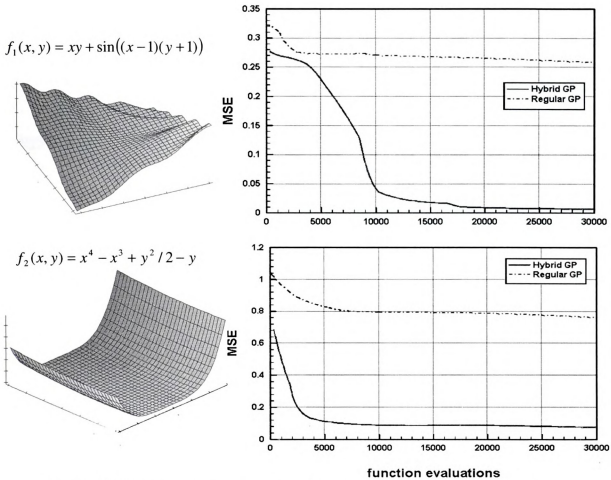


Figure 4.8: Surface fitting test problems f_1, f_2 and respective learning curves

We have focused on the length (number of nodes) and on the number of coefficients in the best evolved programs (remember, that length had an upper limit too). As Table 4.2 illustrates both values are noticeably greater for the programs evolved by HGP. This is one illustration of the inductive bias of the hybrid algorithm. More adaptive programs use more coefficients and consequently have lengthier representations. Also, the number of the terminal inputs (x and y) in HGP results is slightly less. Figure 4.7 shows typical changes in the number of coefficients for a "best" individual on a generational scale for both GP and HGP.

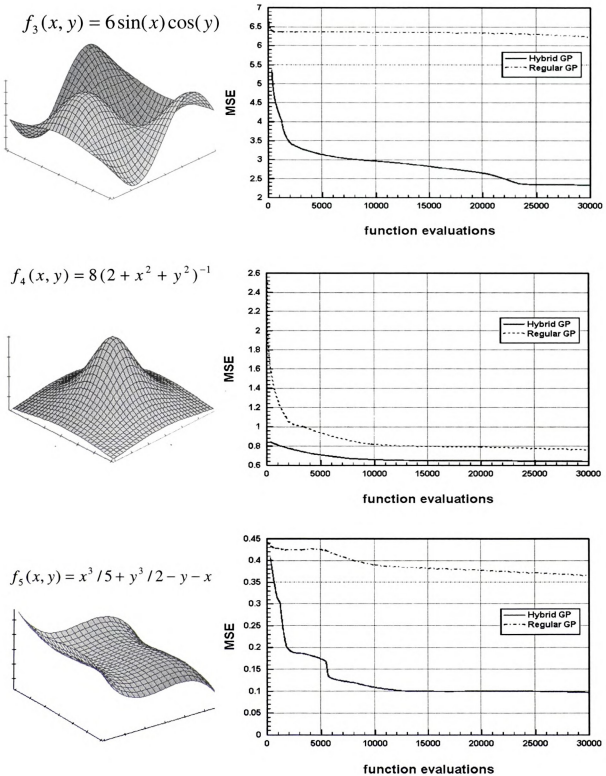


Figure 4.9. Surface fitting test problems f_3, f_4, f_5 and respective learning curves.

4.8 Conclusions

We examined the effectiveness of gradient search optimization of numeric leaf values for Genetic Programming. Genetic search for tree-like programs at the population level is complemented by the optimization of terminal values at the individual level. Local adaptation of individuals is made easier by algorithmic differentiation. We show how conventional random constants are tuned by gradient descent with minimal overhead. Local learning in the form of gradient descent can be efficiently included into GP search. Several experiments with symbolic regression problems are performed to demonstrate the approach's effectiveness. This learning provides a substantial improvement in both final fitness and speed in reaching this fitness. Effects of local learning are clearly manifest in both improved approximation accuracy and selection changes when periods of local and global search are interleaved. Finally, the inductive bias of local learning is quantified. The use of local learning creates a bias in the structure of the solutions, namely it prefers structures that are more readily adaptable by local learning. This approach can have significant impact on practical, engineering problems that are addressed by GP.

Chapter 5

Clustering Ensembles

In contrast to supervised classification, clustering is an inherently ill-posed problem, whose solution violates at least one of the common assumptions about scale-invariance, richness, and cluster consistency [72]. Different clustering solutions may seem equally plausible without a priori knowledge about the underlying data distributions. Every clustering algorithm implicitly or explicitly assumes a certain data model, and it may produce erroneous or meaningless results when these assumptions are not satisfied by the sample data. Thus the availability of prior information about the data domain is crucial for successful clustering, though such information can be hard to obtain, even from experts. Identification of relevant subspaces [2] or visualization [57] may help to establish the sample data's conformity to the underlying distributions or, at least, to the proper number of clusters.

The exploratory nature of clustering tasks demands efficient methods that would benefit from combining the strengths of many individual clustering algorithms. This is the focus of research on clustering ensembles, seeking a combination of multiple partitions that provides improved overall clustering of the given data. Clustering ensembles can go beyond what is typically achieved by a single clustering algorithm in several respects:

- *Robustness*. Better average performance across the domains and datasets.
- *Novelty*. Finding a combined solution unattainable by any single clustering algorithm.
- *Stability and confidence estimation*. Clustering solutions with lower sensitivity to noise, outliers or sampling variations. Clustering uncertainty can be assessed from ensemble distributions.
- *Parallelization and Scalability*. Parallel clustering of data subsets with subsequent combination of results. Ability to integrate solutions from multiple distributed sources of data or attributes (features).

Clustering ensembles can also be used in multiobjective clustering as a compromise between individual clusterings with conflicting objective functions. Fusion of clusterings using multiple sources of data or features becomes increasingly important in distributed data mining, e.g., see review in [101]. Several recent independent studies [29][35][38][39][109][124] have pioneered clustering ensembles as a new branch in the conventional taxonomy of clustering algorithms [63][65].

The problem of clustering combination can be defined generally as follows: given multiple clusterings of the data set, find a combined clustering with better quality. While the problem of clustering combination bears some traits of a classical clustering problem, it also has three major issues which are specific to combination design:

1. Consensus function: How to combine different clusterings? How to resolve the label correspondence problem? How to ensure symmetrical and unbiased consensus with respect to all the component partitions?

2. Diversity of clustering: How to generate different partitions? What is the source of diversity in the components?

3. Strength of constituents/components: How "weak" could each input partition be? What is the minimal complexity of component clusterings to ensure a successful combination?

Similar questions have already been addressed in the framework of multiple classifier systems. Combining results from many supervised classifiers is an active research area [110][11] and it provides the main motivation for clustering combination. However, it is not possible to mechanically apply the combination algorithms from the classification (supervised) domain to clustering (unsupervised) domain. Indeed, no labeled training data is available in clustering; therefore the ground truth feedback necessary for boosting the overall accuracy cannot be used. In addition, different clusterings may produce incompatible data labelings, resulting in intractable correspondence problems, especially when the numbers of clusters are different. Still, the supervised classifier combination demonstrates, in principle, how multiple outputs reduce the variance component of the expected error rate and increase the robustness of the solution.

From the supervised case we also learn that the proper combination of weak classifiers [11][43][59][71] may achieve arbitrarily low error rates on training data, as well as reduce the predictive error. One can expect that using many simple, but computationally inexpensive components will be preferred to combining clusterings obtained by sophisticated, but computationally involved algorithms.

This research further advances ensemble methods in several aspects, namely, design of new effective consensus functions, development of new partition generation mechanisms and study of their clustering accuracy.

We offer a representation of multiple clusterings as a set of new attributes characterizing the data items. Such a view directly leads to a formulation of the combination problem as a categorical clustering problem in the space of these attributes,

or, in other terms, a median partition problem. Median partition can be viewed as the best summary of the given input partitions. As an optimization problem, median partition is NP-complete [6], with a continuum of heuristics for an approximate solution.

This study focuses on the primary problem of clustering ensembles, namely the consensus function, which creates the combined clustering. We show how median partition is related to the classical intra-class variance criterion when generalized mutual information is used as the evaluation function. A consensus function based on quadratic mutual information (QMI) is proposed and reduced to the *k*-means clustering in the space of specially transformed cluster labels.

We also propose a new fusion method for unsupervised decisions that is based on a probability model of the consensus partition in the space of contributing clusters. The consensus partition is found as a solution to the maximum likelihood problem for a given clustering ensemble. The likelihood function of an ensemble is optimized with respect to the parameters of a finite mixture distribution. Each component in this distribution corresponds to a cluster in the target consensus partition, and is assumed to be a multivariate multinomial distribution. The maximum likelihood problem is solved using the EM algorithm [25].

There are several advantages to QMI and EM consensus functions. These include: (i) complete avoidance of solving the label correspondence problem, (ii) low computational complexity, (iii) ability to handle missing data, i.e., missing cluster labels for certain patterns in the ensemble (for example, when bootstrap method is used to generate the ensemble).

Another goal of our work is to adopt weak clustering algorithms and combine their outputs. Vaguely defined, a weak clustering algorithm produces a partition which is only

slightly better than a random partition of the data. We propose two different weak clustering algorithms as the component generation mechanisms:

1. Clustering of random 1-dimensional projections of multidimensional data. This can be generalized to clustering in any random subspace of the original data space.

2. Clustering by splitting the data using a number of random hyperplanes. For example, if only one hyperplane is used then data is split into two groups.

Finally, this work compares the performance of different consensus functions. We have investigated the performance of a family of consensus functions based on categorical clustering including the co-association-based hierarchical methods [39][40][41], hypergraph algorithms [124] and our new consensus functions. Combination accuracy is analyzed as a function of the number and the resolution of the clustering components. In addition, we study clustering performance when some cluster labels are missing, which is often encountered in the distributed data or re-sampling scenarios.

5.1 Related Work

Approaches to combination of clusterings differ in two main respects, namely the way in which the contributing component clusterings are obtained and the method by which they are combined. Fred [38] proposed to summarize various clustering results in a co-association matrix. Co-association values represent the strength of association between objects by analyzing how often each pair of objects appears in the same cluster. The final clustering in [38] is determined using a voting-type algorithm applied to the co-association matrix. Hence the co-association matrix serves as a similarity matrix for the data items. Clusters are formed from the co-association matrix by linking the objects

whose co-association value exceeds a certain threshold. Further work by Fred and Jain [38][39][40] also used co-association values, but instead of a fixed threshold, they applied a hierarchical (single-link) clustering to the co-association matrix. These studies use the k-means algorithm with various values of k and random initializations (for selecting the k cluster centers) for generating the component clusterings. Specific methods for combination of the k-means algorithm and agglomerative linkage algorithms were proposed in [109].

Strehl and Ghosh [124] have considered three different consensus functions for ensemble clustering: one is similar to the evidence accumulation approach [39][40], and the other two methods are based on a hypergraph representation. All of them use various hypergraph operations to search for a solution. The Cluster-based Similarity Partitioning Algorithm (CSPA) [124] induces a graph from a co-association matrix and clusters it using the METIS algorithm [69]. Hypergraph partitioning represents each cluster by a hyperedge in a graph, where the nodes correspond to a given set of objects. Good hypergraph partitions are found using minimal cut algorithms such as HMETIS [68] coupled with the proper objective functions, which also control partition size. Hyperedge collapsing operations are considered in another hypergraph-based Meta-Clustering algorithm (MCLA) in [124]. The meta-clustering algorithm uses these operations to determine soft cluster-membership values for each object.

A combination of partitions obtained from multiple bootstrap samples is implemented in [29][35]. Both works pursued direct re-labeling approaches to the correspondence problem. A re-labeling can be done optimally between two clusterings using the Hungarian algorithm [103]. This was used in [29] to re-label each bootstrap partition using a single reference partition. The reference partition is determined by a

single clustering of the entire data set. After an overall consistent re-labeling, voting can be applied to determine cluster membership for each pattern.

Bagging of multiple *k*-means clustering results was done in [84] by clustering *k*-means centers and assigning the objects to the closest cluster center. In fact, their component clusterings do not keep information about the individual object labels but only information about cluster prototypes. The *k*-means centers are "bagged" and clustered by a hierarchical procedure. Such an approach is unique in that the components of the combination and the final clustering are defined implicitly via prototypes rather than by explicit labelings [84].

Kellam et al. [70] also combined clusterings through a type of co-association matrix. However, this matrix is used only to find the clusters with the highest value of support based on object co-occurrences. As a result, only a set of so-called robust clusters is produced which may not contain all the initial objects

A genetic algorithm is employed in [46] to produce the most stable partitions from an evolving ensemble (population) of clustering algorithms along with a special objective function. The objective function evaluates multiple partitions according to changes caused by data perturbations and prefers those clusterings that are least susceptible to those perturbations.

Dimitriadou et al. [24] proposed a voting/merging procedure that combines clusterings pair-wise and iteratively. The cluster correspondence problem must be solved at each iteration and the solution is not unique. Fuzzy membership decisions are accumulated during the course of merging. The final clustering is obtained by assigning each object to a derived cluster with the highest membership value.

The distributed clustering algorithm [67] constructs a global dendrogram for a set of objects from multiple local models produced by single-link algorithms. Collective

hierarchical clustering combines dendrograms built on different subsets of features. The global hierarchy uses a specific approximation of distance that is estimated based on merging thresholds of component dendrograms. Recently, Fern and Brodley [34] introduced random projections algorithm in the context of clustering ensembles similar to one of the generation mechanisms considered in this paper. Generally clustering ensembles approaches can be summarized as following:

Generative mechanisms (How to obtain different partitions?)

1. Apply various clustering algorithms [124]
2. Use a single algorithm
   - 2.1. Different initialization and built-in randomness [38][39][40][41]
   - 2.2. Different parameters [39]
   - 2.3. Different subsets of data points
     - Deterministic subsets [41][124]
     - Resampling [29][35][94] without or with replacement (e.g., bootstrap)
   - 2.4. Projecting data onto different subspaces [34] (and this study)
   - 2.5. Different subset of features [124]

Consensus functions (How to integrate multiple partitions?)

1. Using Co-association Matrix [38][39][40][41]
   - Single Link (SL)/ Minimum Spanning Tree (MST)
   - Complete Link (CL)
   - Average Link (AL)
   - Ward, or other similarity based algorithms
2. (Hyper) Graph Partitioning [124][125]
   - Hyper Graph Partition Algorithm (HGPA)
   - Meta Clustering Algorithm (MCLA)
   - Clustering Similarity Partition Algorithm (CSPA)
3. Information-theoretic methods, e.g. Quadratic Mutual Information (this study)
4. Voting Approach [29][35]
5. Mixture Model (this study)

Unique properties of the space of cluster labels are exploited by several known consensus functions that we briefly review.

Co-association Methods. Similarity between objects can be estimated by the number of clusters shared by two objects in all the partitions of an ensemble. This similarity definition expresses the strength of co-association of objects by a matrix containing the values:

$$S_{ij} = S(x_i, x_j) = \frac{1}{H}\sum_{k=1}^{H}\delta(\pi_k(x_i), \pi_k(x_j)), \tag{5.1}$$

$$\delta(a,b) \equiv \begin{cases} 1, & \text{if } a = b \\ 0, & \text{if } a \neq b. \end{cases}$$

Thus, one can use numerous similarity-based clustering algorithms by applying them to the matrix of co-association values. It is also possible to construct more sophisticated similarity definitions when the ensemble Π contains "soft" partitions. There are three main concerns with this intuitively appealing approach. First, it has a quadratic complexity in the number of patterns $O(N^2)$. Second, there are no established guidelines concerning which clustering algorithm should be applied, e.g. single linkage or complete linkage. The shapes of the clusters embedded in the space of clustering labels may or may not reflect cluster shapes in the original space. Third, an ensemble with a small number of clusterings may not provide a reliable estimate of the co-association values.

Hypergraph Methods. All the clusters in the ensemble partitions can be represented as hyperedges on a graph with N vertices. Each hyperedge describes a set of objects belonging to the same cluster. A consensus function is formulated as a solution to k-way min-cut hypergraph partitioning problem. Each connected component after the cut corresponds to a cluster in the consensus partition. Though the hypergraph partitioning problem is NP-hard, efficient heuristics have been developed for its solution. However,

the cuts only remove hyperedges as a whole. Hypergraph algorithms seem to work the best for nearly balanced clusters. For example, complexity of CSPA, HGPA and MCLA is estimated in [124] as $O(kN^2H)$, $O(kNH)$, and $O(k^2NH^2)$, respectively

Re-labeling and Voting Approach. If a label correspondence problem is solved for all the given partitions, then a simple voting procedure can be used to assign objects in clusters. However, label correspondence is exactly what makes unsupervised combination difficult. A heuristic approximation to consistent labeling is possible. All the partitions in the ensemble can be re-labeled according to their best agreement with some chosen reference partition. The reference partition can be taken as one from the ensemble, or from a new clustering of the dataset. Also, a meaningful voting procedure assumes that the number of clusters in every given partition is the same as in the target partition. But this requires that the number of clusters in the target consensus partition is known.

To summarize, existing consensus functions suffer from a number of drawbacks that include complexity, heuristic character of objective function and uncertain statistical status of the consensus solution. The next sections introduce new models of the clustering combination that aim to overcome these drawbacks.

5.2 Representation of Multiple Partitions

Combination of multiple partitions can be viewed as a partitioning task itself. Typically, each partition in the combination is represented as a set of labels assigned by a clustering algorithm. The combined partition is obtained as a result of yet another clustering algorithm whose inputs are the cluster labels of the contributing partitions. We will

assume that the labels are nominal values. In general, the clusterings can be "soft", i.e., described by the real values indicating the degree of pattern membership in each cluster in a partition. We consider only "hard" partitions below, noting, however, that combination of "soft" partitions can be solved by numerous clustering algorithms and does not appear to be more complex.

Suppose we are given a set of N data points $X = \{\mathbf{x}_1,\ldots, \mathbf{x}_N\}$ and a set of H partitions $\Pi=\{\pi_1,\ldots, \pi_H\}$ of objects in X. Different partitions of X return a set of labels for each point $\mathbf{x}_i$, $i=1,\ldots, N$:

$$\mathbf{x}_i \rightarrow \{\pi_1(\mathbf{x}_i), \pi_2(\mathbf{x}_i),\ldots,\pi_H(\mathbf{x}_i)\}.$$

Here, H different clusterings are indicated and $\pi_j(\mathbf{x}_i)$ denotes a label assigned to $\mathbf{x}_i$ by the j-th algorithm. No assumption is made about the correspondence between the labels produced by different clustering algorithms. Also no assumptions are needed at the moment about the data input: it could be represented in a non-metric space or as an $N\times N$ dissimilarity matrix. For simplicity, we use the notation $y_{ij} = \pi_j(\mathbf{x}_i)$ or $\mathbf{y}_i = \pi(\mathbf{x}_i)$. The problem of clustering combination is to find a new partition π_C of data X that summarizes the information from the gathered partitions Π. Our main goal is to construct a consensus partition without the assistance of the original patterns in X, but only from their labels Y delivered by the contributing clustering algorithms. Thus, such potentially important issues as the underlying structure of both the partitions and data are ignored for the sake of a solution to the unsupervised consensus problem. We emphasize that a space of new features is induced by the set Π. One can view each component partition π_i as a new feature with categorical values, i.e. cluster labels. The values assumed by the i-th new feature are simply the cluster labels from partition π_i. Therefore, membership of an object

$\mathbf{x}$ in different partitions is treated as a new feature vector $\mathbf{y} = \pi(\mathbf{x})$, an H-tuple. In this case, one can consider partition $\pi_j(x)$ as a feature extraction function. Combination of clusterings becomes equivalent to the problem of clustering of H-tuples if we use only the existing clusterings $\{\pi_1,\ldots, \pi_H\}$, without the original features of data X. Hence the problem of combining partitions can be transformed to a categorical clustering problem. Such a view gives insight into the properties of the expected combination, which can be inferred through various statistical and information-theoretic techniques. In particular, one can estimate the sensitivity of the combination to the correlation of components (features) as well as analyze various sample size issues. Perhaps the main advantage of this representation is that it facilitates the use of known algorithms for categorical clustering [90][36] and allows one to design new consensus heuristics in a transparent way. The extended representation of data X can be illustrated by a table with N rows and $(d+H)$ columns:

				π_1	...	π_H
$\mathbf{x_1}$	x_{11}	...	x_{1d}	$\pi_1(x_1)$	...	$\pi_H(x_1)$
$\mathbf{x_2}$	x_{21}	...	x_{2d}	$\pi_1(x_2)$	...	$\pi_H(x_2)$
...	...	...	...	...	...	...
$\mathbf{x_N}$	x_{N1}	...	x_{Nd}	$\pi_1(x_N)$	...	$\pi_H(x_N)$
	Original d features			"New" H features		

The consensus clustering is found as a partition π_C of a set of vectors $\boldsymbol{Y} = \{\mathbf{y}_i\}$ that directly translates to the partition of the underlying data points $\{\mathbf{x}_i\}$.

5.3 A Mixture Model of Consensus

Our approach to the consensus problem is based on a finite mixture model for the probability of the cluster labels $\mathbf{y}=\pi(\mathbf{x})$ of the pattern/object $\mathbf{x}$. The main assumption is

that the labels $\mathbf{y}_i$ are modeled as random variables drawn from a probability distribution described as a mixture of multivariate component densities:

$$P(\mathbf{y}_i \mid \Theta) = \sum_{m=1}^{M} \alpha_m P_m(\mathbf{y}_i \mid \boldsymbol{\theta}_m), \tag{5.2}$$

where each component is parametrized by $\boldsymbol{\theta}_m$. The M components in the mixture are identified with the clusters of the consensus partition π_C. The mixing coefficients α_m correspond to the prior probabilities of the clusters. In this model, data points $\{\mathbf{y}_i\}$ are presumed to be generated in two steps: first, by drawing a component according to the probability mass function α_m, and then sampling a point from the distribution $P_m(\mathbf{y}|\theta_m)$. All the data $\boldsymbol{Y} = \{\mathbf{y}_i\}_{i=1}^{N}$ are assumed to be independent and identically distributed. This allows one to represent the log likelihood function for the parameters $\Theta = \{\alpha_1, \ldots, \alpha_M, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M\}$ given the data set $\boldsymbol{Y}$ as:

$$\log L(\Theta \mid \mathbf{Y}) = \log \prod_{i=1}^{N} P(\mathbf{y}_i \mid \Theta) = \sum_{i=1}^{N} \log \sum_{m=1}^{M} \alpha_m P_m(\mathbf{y}_i \mid \boldsymbol{\theta}_m). \tag{5.3}$$

The objective of consensus clustering is now formulated as a maximum likelihood estimation problem. To find the best fitting mixture density for a given data $\boldsymbol{Y}$, we must maximize the likelihood function with respect to the unknown parameters Θ:

$$\Theta^* = \arg\max_{\Theta} \log L(\Theta \mid \mathbf{Y}). \tag{5.4}$$

The next important step is to specify the model of component-conditional densities $P_m(\mathbf{y}|\theta_m)$. Note, that the original problem of clustering in the space of data $\boldsymbol{X}$ has been transformed, with the help of multiple clustering algorithms, to a space of new multivariate features $\mathbf{y} = \pi(\mathbf{x})$. To make the problem more tractable, a conditional independence assumption is made for the components of vector $\mathbf{y}_i$, namely that the conditional probability of $\mathbf{y}_i$ can be represented as the following product:

$$P_m(\mathbf{y}_i|\boldsymbol{\theta}_m) = \prod_{j=1}^{H} P_m^{(j)}(y_{ij}|\boldsymbol{\theta}_m^{(j)}). \quad (5.5)$$

To motivate this, one can note that even if the different clustering algorithms (indexed by j) are not truly independent, the approximation by product in Eq. (5.5) can be justified by the excellent performance of naive Bayes classifiers in discrete domains [34]. Our ultimate goal is to make a discrete label assignment to the data in $\boldsymbol{X}$ through an indirect route of density estimation of $\boldsymbol{Y}$. The assignments of patterns to the clusters in π_C are much less sensitive to the conditional independence approximation than the estimated values of probabilities $P(\mathbf{y}_i|\Theta)$, as supported by the analysis of naïve Bayes classifier in [27].

The last ingredient of the mixture model is the choice of a probability density $P_m^{(j)}(y_{ij}|\boldsymbol{\theta}_m^{(j)})$ for the components of the vectors $\mathbf{y}_i$. Since the variables y_{ij} take on nominal values from a set of cluster labels in the partition π_j, it is natural to view them as the outcome of a multinomial trial:

$$P_m^{(j)}(y|\boldsymbol{\theta}_m^{(j)}) = \prod_{k=1}^{K(j)} \vartheta_{jm}(k)^{\delta(y,k)}. \quad (5.6)$$

Here, without the loss of generality, the labels of the clusters in π_j are chosen to be integers in $\{1,\ldots,K(j)\}$. To clarify the notation, note that the probabilities of the outcomes are defined as $\vartheta_{jm}(k)$ and the product is over all the possible values of y_{ij} labels of the partition π_j. Also, the probabilities sum up to one:

$$\sum_{k=1}^{K(j)} \vartheta_{jm}(k) = 1, \forall j \in \{1,\ldots,H\}, \ \forall m \in \{1,\ldots,M\}. \quad (5.7)$$

For example, if the j-th partition has only two clusters, and possible labels are 0 and 1, then Eq. (5.6) can be simplified as:

$$P^{(j)}(y_i|\boldsymbol{\theta}^{(j)}) = \vartheta^{y} (1-\vartheta)^{1-y} \quad (5.8)$$

The maximum likelihood problem in Eq. (5.3) generally cannot be solved in a closed form when all the parameters $\Theta=\{\alpha_1,\ldots,\alpha_M,\theta_1,\ldots,\theta_M\}$ are unknown. However, the likelihood function in Eq. (5.2) can be optimized using the EM algorithm. In order to adopt the EM algorithm, we hypothesize the existence of hidden data $\boldsymbol{Z}$ and the likelihood of complete data $(\boldsymbol{Y}, \boldsymbol{Z})$. If the value of $\mathbf{z}_i$ is known then one could immediately tell which of the M mixture components was used to generate the point $\mathbf{y}_i$. The detailed derivation of the EM solution to the mixture model with multivariate, multinomial components is given in the Appendix. Here we give only the equations for the E- and M-steps which are repeated at each iteration of the algorithm:

$$E[z_{im}]=\frac{\alpha'_m\prod_{j=1}^{H}\prod_{k=1}^{K(j)}\left(\vartheta'_{jm}(k)\right)^{\delta(y_{ij},k)}}{\sum_{n=1}^{M}\alpha'_n\prod_{j=1}^{H}\prod_{k=1}^{K(j)}\left(\vartheta'_{jn}(k)\right)^{\delta(y_{ij},k)}}. \tag{5.9}$$

$$\alpha_m=\frac{\sum_{i=1}^{N}E[z_{im}]}{\sum_{i=1}^{N}\sum_{m=1}^{M}E[z_{im}]}. \tag{5.10}$$

$$\vartheta_{jm}(k)=\frac{\sum_{i=1}^{N}\delta(y_{ij},k)E[z_{im}]}{\sum_{i=1}^{N}\sum_{k=1}^{K(j)}\delta(y_{ij},k)E[z_{im}]}. \tag{5.11}$$

The solution to the consensus clustering problem is obtained by a simple inspection of the expected values of the variables $E[z_{im}]$, due to the fact that $E[z_{im}]$ represents the probability that the pattern $\mathbf{y}_i$ was generated by the m-th mixture component. Once convergence is achieved, a pattern $\mathbf{y}_i$ is assigned to the component which has the largest value for the hidden label $\mathbf{z}_i$.

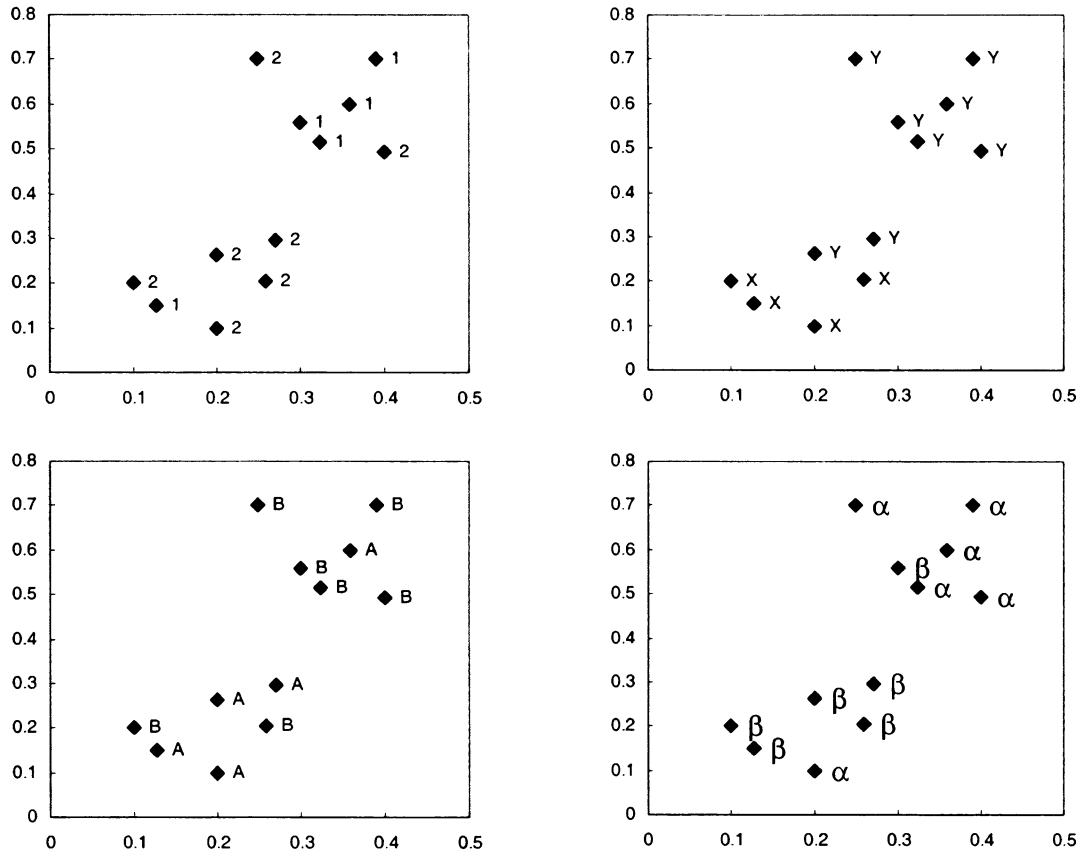


Figure 5.1: Four possible partitions of 12 data points into 2 clusters. Different partitions use different sets of labels.

Table 5.1 : Clustering ensemble and consensus solution

	π_1	π_2	π_3	π_4	$E[z_{i1}]$	$E[z_{i2}]$	Consensus
y_1	2	B	X	β	0.999	0.001	**1**
y_2	2	A	X	α	0.997	0.003	**1**
y_3	2	A	Y	β	0.943	0.057	**1**
y_4	2	B	X	β	0.999	0.001	**1**
y_5	1	A	X	β	0.999	0.001	**1**
y_6	2	A	Y	β	0.943	0.057	**1**
y_7	2	B	Y	α	0.124	0.876	**2**
y_8	1	B	Y	α	0.019	0.981	**2**
y_9	1	B	Y	β	0.260	0.740	**2**
y_{10}	1	A	Y	α	0.115	0.885	**2**
y_{11}	2	B	Y	α	0.124	0.876	**2**
y_{12}	1	B	Y	α	0.019	0.981	**2**

It is instructive to consider a simple example of an ensemble. Figure 5.1 shows four 2-cluster partitions of 12 two-dimensional data points. The correspondence problem is emphasized by the different label systems used by the partitions. Table 5.1 shows the expected values of latent variables after 6 iterations of the EM algorithm and the resulting consensus clustering. In fact, a stable combination appears as early as the third iteration, and it corresponds to the true underlying structure of the data.

Our mixture model of consensus admits generalization for clustering ensembles with incomplete partitions. Such partitions can appear as a result of clustering of subsamples or resampling of a dataset. For example, a partition of a bootstrap sample only provides labels for the selected points. Therefore, the ensemble of such partitions is represented by a set of vectors of cluster labels with potentially missing components. Moreover, different vectors of cluster labels are likely to miss different components. Incomplete information can also arise when some clustering algorithms do not assign outliers to any of the clusters. Different clusterings in the diverse ensemble can consider the same point $\mathbf{x}_i$ as an outlier or otherwise, that results in missing components in the vector $\mathbf{y}_i$. Yet another scenario leading to missing information can occur in clustering combination of distributed data or ensemble of clusterings of non-identical replicas of a dataset.

It is possible to apply the EM algorithm in the case of missing data [47], namely missing cluster labels for some of the data points. In these situations, each vector $\mathbf{y}_i$ in $\boldsymbol{Y}$ can be split into observed and missing components $\mathbf{y}_i = (\mathbf{y}_i^{obs}, \mathbf{y}_i^{mis})$. Incorporation of a missing data leads to a slight modification of the computation of E and M steps. First, the expected values $E[z_{im} | \mathbf{y}_i^{obs}, \Theta']$ are now inferred from the observed components of vector

$\mathbf{y}_i$, i.e. the products in Eq. (5.9) are taken over known labels: $\prod_{j=1}^{H} \rightarrow \prod_{j:y^{obs}}$. Additionally, one must compute the expected values $E[z_{im}\, \mathbf{y}_i^{mis} | \mathbf{y}_i^{obs}, \Theta']$ and substitute them, as well as $E[z_{im} | \mathbf{y}_i^{obs}, \Theta']$, in the M-step for re-estimation of parameters $\vartheta_{jm}(k)$. More details on handling missing data can be found in [47].

Though data with missing cluster labels can be obtained in different ways, we analyze only the case when components of $\mathbf{y}_i$ are missing completely at random [116]. It means that the probability of a component to be missing does not depend on other observed or unobserved variables. Note, that the outcome of clustering of data subsamples (e.g., bootstrap) is different from clustering the entire data set and then deleting a random subset of labels. However, our goal is to present a consensus function for general settings. We expect that experimental results for ensembles with missing labels are applicable, at least qualitatively, even for a combination of bootstrap clusterings.

The proposed ensemble clustering based on mixture model consensus algorithm is summarized below. Note that any clustering algorithm can be used to generate the ensemble instead of the *k*-means algorithm shown in this pseudocode:

---

begin

for i=1 **to** H // H - number of clusterings

cluster a dataset $\pi \leftarrow$ *k*-means(X)

add partition to the ensemble $\Pi = \{\Pi, \pi\}$

end

initialize model parameters $\Theta = \{\alpha_1, \ldots, \alpha_M, \theta_1, \ldots, \theta_M\}$

do until convergence criterion is satisfied

compute expected values $E[z_{im}]$, i=1..N, m=1..M

compute $E[z_{im}\ \mathbf{y}_i^{mis}]$ for missing data (if any)

re-estimate parameters $\vartheta_{jm}(k)$, j=1..H, m=1..M, $\forall k$

end

$\pi_C(\mathbf{x}_i)$ = index of component of $\mathbf{z}_i$ with the largest expected value, i=1..N

return π_C // consensus partition

end

The value of M, number of components in the mixture, deserves a separate discussion that is beyond the scope of this paper. Here, we assume that the target number of clusters is predetermined. It should be noted, however, that the mixture model in unsupervised classification greatly facilitates estimation of the true number of clusters [37]. Maximum likelihood formulation of the problem specifically allows us to estimate M by using additional objective functions during the inference, such as the minimum description length of the model. In addition, the proposed consensus algorithm can be viewed as a version of Latent Class Analysis (e.g. see [7]), which has rigorous statistical means for quantifying plausibility of a candidate mixture model.

Whereas the finite mixture model may not be valid for the patterns in the original space (the initial representation), this model more naturally explains the separation of groups of patterns in the space of "extracted" features (labels generated by the partitions). It is somewhat reminiscent of classification approaches based on kernel methods which rely on linear discriminant functions in the transformed space. For example, Support Vector Clustering [9] seeks spherical clusters after the kernel transformation that correspond to more complex cluster shapes in the original pattern space.

5.4 Information-theoretic consensus of clusterings

Another candidate consensus function is based on the notion of median partition. A median partition σ is the best summary of existing partitions in Π. In contrast to the co-association approach, the median partition is derived from estimates of similarities between attributes (i.e., partitions in Π), rather than from similarities between objects. A well-known example of this approach is implemented in the COBWEB algorithm in the context of conceptual clustering [36]. The COBWEB clustering criterion estimates the partition utility, which is the sum of category utility functions introduced by Gluck and Corter [48]. In our terms, the category utility function $U(\sigma, \pi_i)$ evaluates the quality of a candidate median partition $\pi_C = \{C_1,\ldots,C_K\}$ against some other partition $\pi_i = \{L^i_1,\ldots, L^i_{K(i)}\}$, with labels L^i_j for j-th cluster:

$$U(\pi_C,\pi_i) = \sum_{r=1}^{K} p(C_r) \sum_{j=1}^{K(i)} p(L^i_j \mid C_r)^2 - \sum_{j=1}^{K(i)} p(L^i_j)^2 , \qquad (5.12)$$

with the following notations: $p(C_r) = |C_r| \;/\; N$, $p(L^i_j) = |L^i_j| \;/\; N$, and $p(L^i_j \mid C_r) = |L^i_j \cap C_r| / |C_r|$.

The function $U(\pi_C, \pi_i)$ assesses the agreement between two partitions as the difference between the expected number of labels of partition π_i that can be correctly predicted both with the knowledge of clustering π_C and without it. The category utility function can also be written as the Goodman-Kruskal index for the contingency table between two partitions [49][93]. The overall utility of the partition π_C with respect to all the partitions in Π can be measured as the sum of pair-wise agreements:

$$U(\pi_C, \Pi) = \sum_{i=1}^{H} U(\pi_C, \pi_i) . \tag{5.13}$$

Therefore, the best median partition should maximize the value of overall utility:

$$\pi_C^{\text{best}} = \arg\max_{\pi_C} U(\pi_C, \Pi) . \tag{5.14}$$

Importantly, Mirkin [93] has proved that maximization of partition utility in Eq. (5.13) is equivalent to minimization of the square-error clustering criterion if the number of clusters K in target partition π_C is fixed. This is somewhat suprising in that the partition utility function in Eq. (5.14) uses only the between-attribute similarity measure of Eq. (5.12), while the square-error criterion makes use of distances between objects and prototypes. Simple standardization of categorical labels in $\{\pi_1,\ldots,\pi_H\}$ effectively transforms them to quantitative features [93]. This allows us to compute real-valued distances and cluster centers. This transformation replaces the i-th partition π_i assuming $K(i)$ values by $K(i)$ binary features, and standardizes each binary feature to a zero mean. In other words, for each object x we can compute the values of the new features $\tilde{y}_{ij}(x)$, as follows:

$$\tilde{y}_{ij}(x) = \delta(L_j^i, \pi_i(x)) - p(L_j^i) , \text{ for } j=1\ldots K(i),\ i=1\ldots H . \tag{5.15}$$

Hence, the solution of median partition problem in Eq. (5.14) can be approached by the k-means clustering algorithm operating in the space of features $\tilde{y}_{ij}$ if the number of target clusters is predetermined. We use this heuristic as a part of empirical study of consensus functions.

Let us consider the information-theoretic approach to the median partition problem. In this framework, the quality of the consensus partition π_C is determined by the

amount of information $I(\pi_C,\Pi)$ it shares with the given partitions in Π. Strehl and Ghosh [124] suggest an objective function that is based on the classical Shannon definition of mutual information:

$$\pi_C^{\text{best}} = \arg\max_{\pi_C} I(\pi_C,\Pi) \text{, where } I(\pi_C,\Pi) = \sum_{i=1}^{H} I(\pi_C,\pi_i), \tag{5.16}$$

$$I(\pi_C,\pi_i) = \sum_{r=1}^{K}\sum_{j=1}^{K(i)} p(C_r,L_j^i)\log\left(\frac{p(C_r,L_j^i)}{p(C_r)p(L_j^i)}\right). \tag{5.17}$$

Again, an optimal median partition can be found by solving this optimization problem. However, it is not clear how to directly use these equations in a search for consensus.

We show that another information-theoretic definition of entropy will reduce the mutual information criterion to the category utility function discussed before. We proceed from the generalized entropy of degree s for a discrete probability distribution $P=(p_1,\ldots,p_n)$ [56]:

$$H^s(P) = (2^{1-s}-1)^{-1}\left(\sum_{i=1}^{n} p_i^s - 1\right), \quad s>0, \; s\neq 1 \tag{5.18}$$

Shannon's entropy is the limit form of Eq.(5.18):

$$\lim_{s\to 1} H^s(P) = -\sum_{i=1}^{n} p_i \log p_i \; . \tag{5.19}$$

Generalized mutual information between σ and π can be defined as:

$$I^s(\pi,\pi_C) = H^s(\pi) - H^s(\pi \mid \pi_C). \tag{5.20}$$

Quadratic entropy ($s = 2$) is of particular interest, since it is known to be closely related to classification error, when used in the probabilistic measure of inter-class distance. When $s = 2$, generalized mutual information $I(\pi_C, \pi_i)$ becomes:

$$I^2(\pi_C,\pi_i) = -2\left(\sum_{j=1}^{K(i)} p(L_j^i)^2 - 1\right) + 2\sum_{r=1}^{K} p(C_r)\left(\sum_{j=1}^{K(i)} p(L_j^i \mid C_r)^2 - 1\right) = \tag{5.21}$$

$$= 2\sum_{r=1}^{K} p(C_r)\sum_{j=1}^{K(i)} p(L_j^i \mid C_r)^2 - 2\sum_{j=1}^{K(i)} p(L_j^i)^2 = 2U(\pi_C,\pi_i).$$

Therefore, generalized mutual information gives the same consensus clustering criterion as the category utility function in Eq. (5.13). Moreover, the traditional Gini-index measure for attribute selection also follows from Eqs. (5.12) and (5.21). In light of Mirkin's result, all these criteria are equivalent to within-cluster variance minimization, after simple label transformation. Quadratic mutual information, mixture model and other interesting consensus functions have been used in our comparative empirical study.

5.5 Combination of Weak Clusterings

The previous sections addressed the problem of clusterings combination, namely how to formulate the consensus function regardless of the nature of individual partitions in the combination. We now turn to the issue of generating different clusterings for the combination. There are several principal questions. Do we use the partitions produced by numerous clustering algorithms available in the literature? Can we relax the requirements for the clustering components? There are several existing methods to provide diverse partitions:

1. Use different clustering algorithms, e.g. *k*-means, mixture of Gaussians, spectral, single-link, etc. [124].

2. Exploit built-in randomness or different parameters of some algorithms, e.g. initializations and various values of k in *k*-means algorithm [39][40][84].

3. Use many subsamples of the data set, such as bootstrap samples [29][91].

These methods rely on the clustering algorithms, which are powerful on their own, and as such are computationally involved. We argue that it is possible to generate the partitions using weak, but less expensive, clustering algorithms and still achieve comparable or better performance. Certainly, the key motivation is that the synergy of many such components will compensate for their weaknesses. We consider two simple clustering algorithms:

1. Clustering of the data projected to a random subspace. In the simplest case, the data is projected on 1-dimensional subspace, a random line. The k-means algorithm clusters the projected data and gives a partition for the combination.

2. Random splitting of data by hyperplanes. For example, a single random hyperplane would create a rather trivial clustering of d-dimensional data by cutting the hypervolume into two regions.

We will show that both approaches are capable of producing high quality consensus clusterings in conjunction with a proper consensus function.

5.5.1 Splitting by Random Hyperplanes

Direct clustering by use of a random hyperplane illustrates how a reliable consensus emerges from low-informative components. The random splits approach pushes the notion of weak clustering almost to an extreme. The data set is cut by random hyperplanes dissecting the original volume of d-dimensional space containing the points. Points separated by the hyperplanes are declared to be in different clusters. Hence, the output clusters are convex. In this situation, a co-association consensus function is

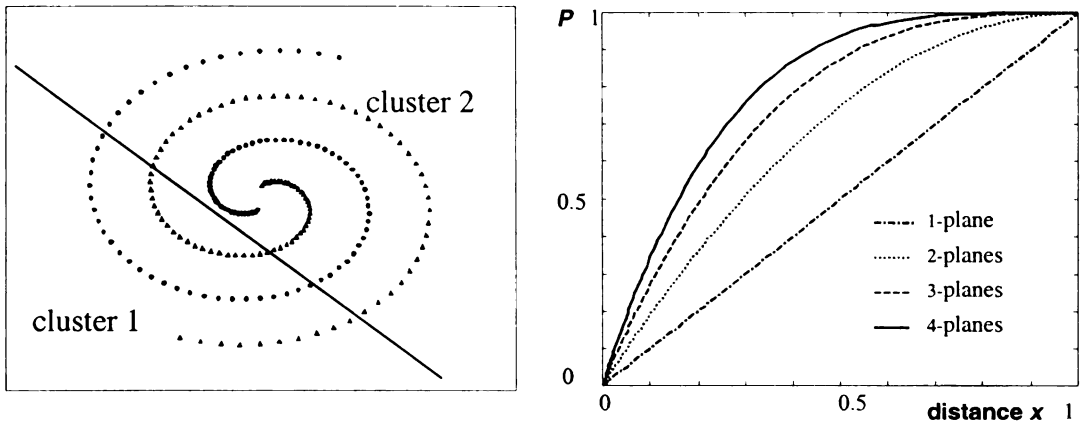


Figure 5.2. Clustering by a random hyperplane: (a) An example of splitting 2-spiral data set by a random line. Points on the same side of the line are in the same cluster. (b) Probability of splitting two one-dimensional objects for different number of random thresholds as a function of distance between objects.

appropriate since the only information needed is whether the patterns are in the same cluster or not. Thus the contribution of a hyperplane partition to the co-association value for any pair of objects can be either 0 or 1. Finer resolutions of distance are possible by counting the number of hyperplanes separating the objects, but for simplicity we do not use it here. Consider a random line dissecting the classic 2-spiral data shown in Figure 5.2(a). While any one such partition does little to reveal the true underlying clusters, analysis of the hyperplane generating mechanism shows how multiple such partitions can discover the true clusters.

Consider first the case of one-dimensional data. Splitting of objects in 1-dimensional space is done by a random threshold in $\mathbf{R}^1$. In general, if r thresholds are randomly selected, then $(r+1)$ clusters are formed. It is easy to derive that, in 1-dimensional space, the probability of separating two objects whose inter-point distance is x is exactly:

$$P(\text{split}) = 1 - (1 - x/L)^r, \tag{5.22}$$

where L is the length of the interval containing the objects, and r threshold points are drawn at random from uniform distribution on this interval. Figure 5.2(b) illustrates the dependence for L=1 and r=1,2,3,4. If a co-association matrix is used to combine H different partitions, then the expected value of co-association between two objects is $H(1-P(\text{split}))$, that follows from the binomial distribution of the number of splits in H attempts. Therefore, the co-association values found after combining many random split partitions are generally expected to be a non-linear and a monotonic function of respective distances. The situation is similar for multidimensional data, however, the generation of random hyperplanes is a bit more complex. To generate a random hyperplane in d dimensions, we should first draw a random point in the multidimensional region that will serve as a point of origin. Then we randomly choose a unit normal vector $\boldsymbol{u}$ that defines the hyperplane. The two objects characterized by vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ will be in the same cluster if $(\boldsymbol{up})(\boldsymbol{uq})>0$ and will be separated otherwise (here $\boldsymbol{ab}$ denotes a scalar product of $\boldsymbol{a}$ and $\boldsymbol{b}$). If r hyperplanes are generated, then the total probability that two objects remain in the same cluster is just the product of probabilities that each of hyperplanes does not split the objects. Thus we can expect that the law governing the co-association values is close to what is obtained in 1-dimensional space in Eq. (5.22).

Let us compare the actual dependence of co-association values with the function in Eq. (5.22). Figure 5.3 shows the results of experiments with 1000 different partitions by random splits of the Iris data set. The Iris data is 4-dimensional and contains 150 points. There are 11,175 pair-wise distances between the data items. For all the possible pairs of points, each plot in Fig. 3 shows the number of times a pair was split. The observed dependence of the inter-point "distances" derived from the co-association

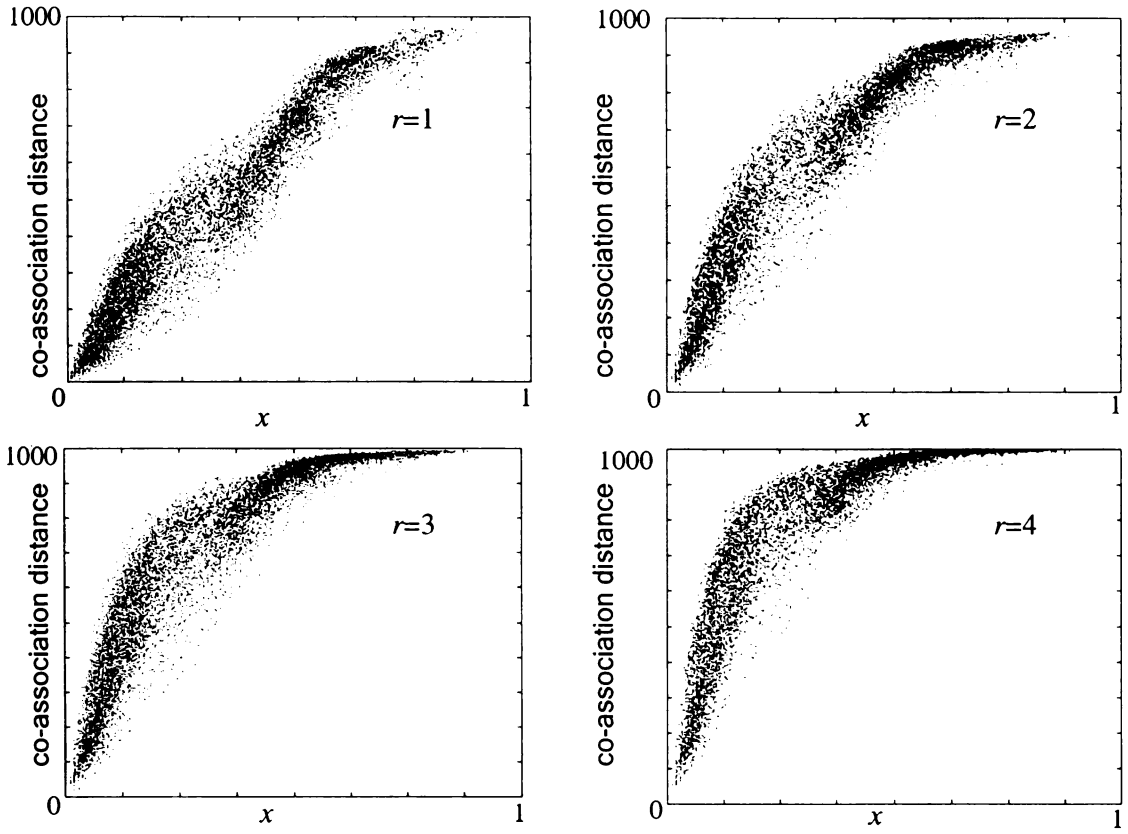


Figure 5.3. Dependence of distances derived from the co-association values vs. the actual Euclidean distance x for each possible pair of objects in Iris data. Co-association matrices were computed for different numbers of hyperplanes r =1,2,3,4.

values vs. the true Euclidean distance, indeed, can be described by the function in Eq. (5.22).

Clearly, the inter-point distances dictate the behavior of respective co-association values. The probability of a cut between any two given objects does not depend on the other objects in the data set. Therefore, we can conclude that any clustering algorithm that works well with the original inter-point distances is also expected to work well with co-association values obtained from combination of multiple partitions by random splits. However, this result is more of theoretical value when true distances are available, since they can be used directly instead of co-association values. It illustrates the main idea of

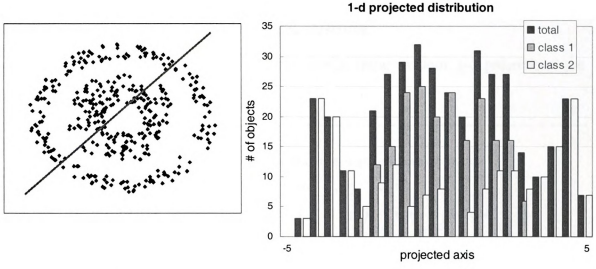
the approach, namely that the synergy of multiple weak clusterings can be very effective. We present an empirical study of the clustering quality of this algorithm in the experimental section.

5.5.2 Combination of Clusterings in Random Subspaces

Random subspaces are an excellent source of clustering diversity that provides different views of the data. Projective clustering is an active topic in data mining. For example, algorithms such as CLIQUE [2] and DOC [105] can discover both useful projections as well as data clusters. Here, however, we are only concerned with the use of random projections for the purpose of clustering combination.

Each random subspace can be of very low dimension and it is by itself somewhat uninformative. On the other hand, clustering in 1-dimensional space is computationally cheap and can be effectively performed by k-means algorithm. The main subroutine of k-means algorithm – distance computation – becomes d times faster in 1-dimensional space. The cost of projection is linear with respect to the sample size and number of dimensions $O(Nd)$, and is less then the cost of one k-means iteration.

The main idea of our approach is to generate multiple partitions by projecting the data on a random line. A fast and simple algorithm such as k-means clusters the projected data, and the resulting partition becomes a component in the combination. Afterwards, a chosen consensus function is applied to the components. We discuss and compare several consensus functions in the experimental section.



(a) **(b)**

Figure 5.4. Projecting data on a random line: (a) A sample data with two identifiable natural clusters and a line randomly selected for projection. (b) Histogram of the distribution of points resulting from data projection onto a random line.

It is instructive to consider a simple 2-dimensional data set and one of its projections, as illustrated in Fig. 5.4(a). There are two natural clusters in the data. This data looks the same in any 1-dimensional projection, but the actual distribution of points is different in different clusters in the projected subspace. For example, Figure 5.4(b) shows one possible histogram distribution of points in 1-dimensional projection of this data. There are three identifiable modes, each having a clear majority of points from one of the two classes. One can expect that clustering by the *k*-means algorithm will reliably separate at least a portion of the points from the outer ring cluster. It is easy to imagine that projection of the data in Figure 5.4(a) on another random line would result in a different distribution of points and different label assignments, but for this particular data set it will always appear as a mixture of three bell-shaped components. Most probably, these modes will be identified as clusters by the *k*-means algorithm. Thus each new 1-dimensional

view correctly helps to group some data points. Accumulation of multiple views eventually should result in a correct combined clustering.

The major steps for combining the clusterings using random 1-d projections are described by the following procedure:

---

begin

  for i=1 **to** H // H is the number of clusterings in the combination

    generate a random vector **u**, s.t. $|\mathbf{u}|=1$

    project all data points $\{\mathbf{x}_j\}$: $\{y_j\} \leftarrow \{\mathbf{u}\mathbf{x}_j\}$, j=1...N

    cluster projections $\{y_j\}$: $\pi(i) \leftarrow k$-means($\{y_j\}$)

  end

combine clusterings via a consensus function: $\sigma \leftarrow \{\pi(i)\}$, i=1...H

return σ // consensus partition

end

The important parameter is the number of clusters in the component partition π_i returned by k-means algorithm at each iteration, i.e. the value of k. If the value of k is too large then the partitions $\{\pi_i\}$ will overfit the data set which in turn may cause unreliability of the co-association values. Too small a number of clusters in $\{\pi_i\}$ may not be enough to capture the true structure of data set. In addition, if the number of clusterings in the combination is too small then the effective sample size for the estimates of distances from co-association values is also insufficient, resulting in a larger variance of the estimates. That is why the consensus functions based on the co-association values are more sensitive to the number of partitions in the combination (value of H) than consensus functions based on hypergraph algorithms.

5.6 Empirical study

The experiments were conducted with artificial and real-world datasets, where true natural clusters are known, to validate both accuracy and robustness of consensus via the mixture model. We explored the datasets using five different consensus functions.

5.6.1 Datasets

Table 5.2 summarizes the details of the datasets. Five datasets of different nature have been used in the experiments. "Biochemical" and "Galaxy" data sets are described in [1] and [99], respectively.

Table 5.2. Characteristics of the datasets.

Dataset	No. of features	No. of classes	No. of points/class	Total no. of points	Av. k-means error (%)
Biochem.	7	2	2138-3404	5542	47.4
Galaxy	14	2	2082-2110	4192	21.1
2-spirals	2	2	100-100	200	43.5
Half-rings	2	2	100-300	400	25.6
Iris	4	3	50-50-50	150	15.1

We evaluated the performance of the evidence accumulation clustering algorithms by matching the detected and the known partitions of the datasets. The best possible matching of clusters provides a measure of performance expressed as the misassignment rate. To determine the clustering error, one needs to solve the correspondence problem between the labels of known and derived clusters. The optimal correspondence can be obtained using the Hungarian method for minimal weight bipartite matching problem with $O(k^3)$ complexity for k clusters.

5.6.2 Selection of Parameters and Algorithms

Accuracy of the QMI and EM consensus algorithms has been compared to six other consensus functions:

1. CSPA for partitioning of hypergraphs induced from the co-association values. Its complexity is $O(N^2)$ that leads to severe computational limitations. We did not apply this algorithm to "Galaxy" [99] and "Biochemical" [1] data. For the same reason, we did not use other co-association methods, such as single-link clustering. The performance of these methods was already analyzed in [38][39].
2. HGPA for hypergraph partitioning.
3. MCLA, that modifies HGPA via extended set of hyperedge operations and additional heuristics.
4. Consensus functions operated on the co-association matrix, but with three different hierarchical clustering algorithms for obtaining the final partition, namely single-linkage, average-linkage, and complete-linkage.

First three methods (CSPA, HGPA and MCLA) were introduced in [47] and their code is available at http://www.strehl.com.

The k-means algorithm was used as a method of generating the partitions for the combination. Diversity of the partitions is ensured by the solutions obtained after a random initialization of the algorithm. The following parameters of the clustering ensemble are especially important:

i. H – the number of combined clusterings. We varied this value in the range [5..50].
ii. k – the number of clusters in the component clusterings $\{\pi_1,\ldots,\pi_H\}$ produced by k-means algorithm was taken in the range [2..10].

iii. r – the number of hyperplanes used for obtaining clusterings $\{\pi_1,\ldots,\pi_H\}$ by random splitting algorithm.

Both, the EM and QMI algorithm are susceptible to the presence of local minima of the objective functions. To reduce the risk of convergence to a lower quality solution, we used a simple heuristic afforded by low computational complexities of these algorithms. The final partition was picked from the results of three runs (with random initializations) according to the value of objective function. The highest value of the likelihood function served as a criterion for the EM algorithm and within-cluster variance is a criterion for the QMI algorithm.

5.6.3 Experiments with Complete Partitions

Main results for each of the data sets are presented in Tables 5.3-7. The tables report the mean error rate (%) of clustering combination from 10 independent runs for the relatively large biochemical and astronomical data sets and from 20 runs for the other smaller data sets.

First observation is that none of the consensus functions is the absolute winner. Good performance was achieved by different combination algorithms across the values of parameters k and H. The EM algorithm slightly outperforms other algorithms for ensembles of smaller size, while MCLA is superior when number of clusterings $H > 20$. However, ensembles of very large size are less important in practice. All co-association methods are usually unreliable with number of clusterings $H < 50$ and this is where we position the proposed EM algorithm. Both, EM and QMI consensus functions need to estimate at least kHM parameters. Therefore, accuracy degradation will inevitably occur with an increase in the number of partitions when sample size is fixed. However, there was no noticeable decrease in the accuracy of the EM algorithm in current experiments.

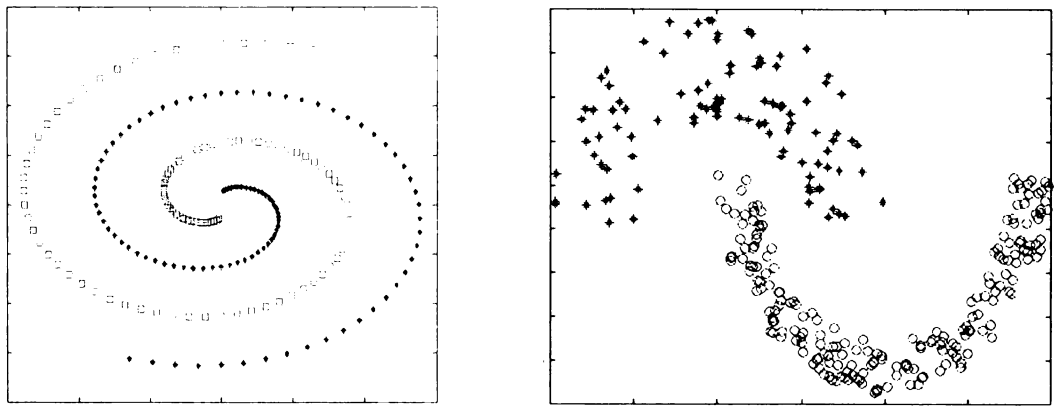


Figure 5.5: "2 spirals" and "Half-rings" datasets difficult for any centroid-based clustering algorithms.

The EM algorithm also should benefit from the datasets of large size due to the improved reliability of model parameter estimation.

A valuable property of the EM consensus algorithm is its fast convergence rate. Mixture model parameter estimates nearly always converged in less than 10 iterations for all the datasets. Moreover, pattern assignments were typically settled in 4-6 iterations.

Clustering combination accuracy also depends on the number of clusters M in the ensemble partitions, or more precisely, on its ratio to the target number of clusters, i.e. k/M. For example, the EM algorithm worked best with k=3 for Iris dataset, k=3,4 for "Galaxy" dataset and k=2 for "Half-rings" data. These values of k are equal or slightly greater than the number of clusters in the combined partition. In contrast, accuracy of MCLA slightly improves with an increase in the number of clusters in the ensemble. Figure 5.7 shows the error as a function of k for different consensus functions for the galaxy data.

It is also interesting to note that, as expected, the average error of consensus clustering was lower than average error of the k-means clusterings in the ensemble (Table 5.2) when

k is chosen to be equal to the true number of clusters. Moreover, the clustering error obtained by EM and MCLA algorithms with k=4 for "Biochemistry" data [1] was the same as found by supervised classifiers applied to this dataset [112].

5.6.4 Experiments with incomplete partitions

This set of experiments focused on the dependence of clustering accuracy on the number of patterns with missing cluster labels. As before, an ensemble of partitions was generated using the k-means algorithm. Then, we randomly deleted cluster labels for a fixed number of patterns in each of the partitions. The EM consensus algorithm was used on such an ensemble. The number of missing labels in each partition was varied between 10% to 50% of the total number of patterns. The main results averaged over 10 independent runs are reported in Table 5.8 for "Galaxy" and "Biochemistry" datasets for various values of H and k. Also, a typical dependence of error on the number of patterns with missing data is shown for Iris data on Figure 5.6 (H=5, k=3).

One can note that combination accuracy decreases only insignificantly for "Biochemistry" data when up to 50% of labels are missing. This can be explained by the low inherent accuracy for this data, leaving little room for further degradation. For the "Galaxy" data, the accuracy drops by almost 10% when k=3,4. However, when just 10-20% of the cluster labels are missing, then there is just a small change in accuracy. Also, with different values of k, we see different sensitivity of the results to the missing labels. For example, with k=2, the accuracy drops by only slightly more than 1%. Ensembles of larger size H=10 suffered less from missing data than ensembles of size H=5.

Table 5.3: Mean error rate (%) for the "Galaxy" dataset.

		Type of Consensus Function			
H	*k*	EM	QMI	HGPA	MCLA
5	2	18.9	19.0	50.0	18.9
5	3	11.6	13.0	50.0	13.5
5	4	11.3	13.0	50.0	11.7
5	5	13.9	18.0	50.0	14.3
5	7	14.5	21.9	50.0	15.6
5	10	13.4	31.1	50.0	15.4
10	2	18.8	18.8	50.0	18.8
10	3	14.9	15.0	50.0	14.8
10	4	11.6	11.1	50.0	12.0
10	5	14.5	13.0	50.0	13.6
15	2	18.8	18.8	50.0	18.8
15	3	14.0	13.3	50.0	14.8
15	4	11.7	11.5	50.0	11.6
15	5	12.9	11.5	50.0	12.9
20	2	18.8	18.9	50.0	18.8
20	3	12.8	11.7	50.0	14.3
20	4	11.0	10.8	50.0	11.5
20	5	16.2	12.1	50.0	12.3

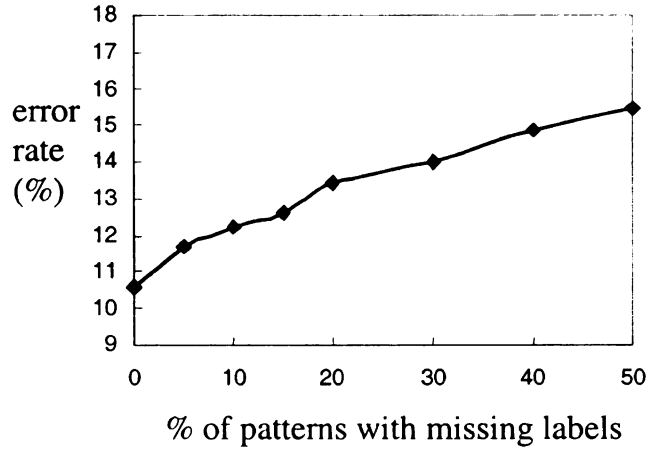


Figure 5.6: Consensus clustering error rate as a function of the number of missing labels in the ensemble for the Iris dataset, H=5, k=3.

5.6.5 Results of random subspaces algorithm

Let us start by demonstrating how the combination of clusterings in projected 1-dimensional subspaces outperforms the combination of clusterings in the original multidimensional space. Figure. 5.8(a) shows the learning dynamics for Iris data and k=4,

using average-link consensus function based on co-association values. Note that the number of clusters in each of the components $\{\pi_1,\ldots, \pi_H\}$ is set to k=4, and is different from the true number of clusters (=3). Clearly, each individual clustering in full multidimensional space is much stronger than any 1-dim partition, and therefore with only a small number of partitions (H<50) the combination of weaker partitions is not yet effective. However, for larger numbers of combined partitions (H>50), 1-dim projections together better reveal the true structure of the data. It is quite unexpected, since the k-means algorithm with k=3 makes, on average, 19 mistakes in the original 4-dimensional space and 25 mistakes in 1-dimensional random subspace. Moreover, clustering in the projected subspace is d times faster than in multidimensional space. However, the cost of computing a consensus partition σ is the same in both cases.

The results regarding the impact of value of k are reported in Figure 5.8(b), which shows that there is a critical value of k for the Iris data set. This occurs when the average-linkage of co-association distances is used as a consensus function. In this case the value k=2 is not enough to separate the true clusters.

Table 5.4: Mean error rate (%) for the "Half-rings"

		Type of Consensus Function				
H	k	**EM**	**QMI**	**CSPA**	**HGPA**	**MCLA**
5	2	25.4	25.4	25.5	50.0	25.4
5	3	24.0	36.8	26.2	48.8	25.1
10	2	26.7	33.2	28.6	50.0	23.7
10	3	33.5	39.7	24.9	26.0	24.2
30	2	26.9	40.6	26.2	50.0	26.0
30	3	29.3	35.9	26.2	27.5	26.2
50	2	27.2	32.3	29.5	50.0	21.1
50	3	28.8	35.3	25.0	24.8	24.6

Table 5.5: Mean error rate (%) for the "Biochemistry" dataset.

		Type of Consensus Function		
H	*k*	**EM**	**QMI**	**MCLA**
5	2	44.8	44.8	44.8
5	3	43.2	48.8	44.7
5	4	42.0	45.6	42.7
5	5	42.7	44.3	46.3
10	2	45.0	45.1	45.1
10	3	44.3	45.4	40.2
10	4	39.3	45.1	37.3
10	5	40.6	45.0	41.2
20	2	45.1	45.2	45.1
20	3	46.6	47.4	42.0
20	4	37.2	42.6	39.8
20	5	40.5	42.1	39.9
30	2	45.3	45.3	45.3
30	3	47.1	48.3	46.8
30	4	37.3	42.3	42.8
30	5	39.9	42.9	38.4
50	2	45.2	45.3	45.2
50	3	46.9	48.3	44.6
50	4	40.1	39.7	42.8
50	5	39.4	38.1	42.1

Table 5.6: Mean error rate (%) for the "2-spirals"

		Type of Consensus Function				
H	*k*	**EM**	**QMI**	**CSPA**	**HGPA**	**MCLA**
5	2	43.5	43.6	43.9	50.0	43.8
5	3	41.1	41.3	39.9	49.5	40.5
5	5	41.2	41.0	40.0	43.0	40.0
5	7	45.9	45.4	45.4	42.4	43.7
5	10	47.3	45.4	47.7	46.4	43.9
10	2	43.4	43.7	44.0	50.0	43.9
10	3	36.9	40.0	39.0	49.2	41.7
10	5	38.6	39.4	38.3	40.6	38.9
10	7	46.7	46.7	46.2	43.0	45.7
10	10	46.7	45.6	47.7	47.1	42.4
20	2	43.3	43.6	43.8	50.0	43.9
20	3	40.7	40.2	37.1	49.3	40.0
20	5	38.6	39.5	38.2	40.0	38.1
20	7	45.9	47.6	46.7	44.4	44.2
20	10	48.2	47.2	48.7	47.3	42.2

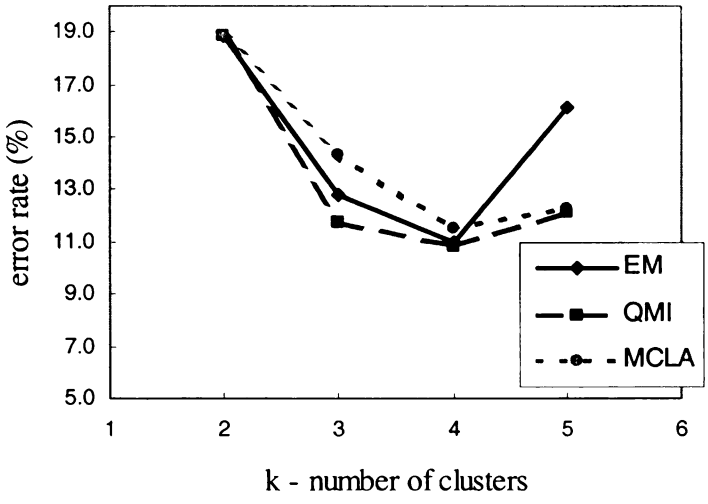


Figure 5.7: Consensus error as a function of the number of clusters in the contributing partitions for Galaxy data and ensemble size H=20.

Table 5.7: Mean error rate (%) for the Iris

		Type of Consensus Function				
H	k	**EM**	**QMI**	**CSPA**	**HGPA**	**MCLA**
5	3	11.0	14.7	11.2	41.4	10.9
10	3	10.8	10.8	11.3	38.2	10.9
15	3	10.9	11.9	9.8	42.8	11.1
20	3	10.9	14.5	9.8	39.1	10.9
30	3	10.9	12.8	7.9	43.4	11.3
40	3	11.0	12.4	7.7	41.9	11.1
50	3	10.9	13.8	7.9	42.7	11.2

Table 5.8: Clustering error rate of EM algorithm as a function of the number of missing labels for the large datasets

H	k	Missing labels (%)	"Galaxy" error (%)	"Biochem." error (%)
5	2	10	18.81	45.18
5	2	20	18.94	44.73
5	2	30	19.05	45.08
5	2	40	19.44	45.64
5	2	50	19.86	46.23
5	3	10	12.95	43.79
5	3	20	13.78	43.89
5	3	30	14.92	45.67
5	3	40	19.58	47.88
5	3	50	23.31	48.41
5	4	10	11.56	43.10
5	4	20	11.98	43.59
5	4	30	14.36	44.50
5	4	40	17.34	45.12
5	4	50	24.47	45.62
10	2	10	18.87	45.14
10	2	20	18.85	45.26
10	2	30	18.86	45.28
10	2	40	18.93	45.13
10	2	50	19.85	45.35
10	3	10	13.44	44.97
10	3	20	14.46	45.20
10	3	30	14.69	47.91
10	3	40	14.40	47.21
10	3	50	15.65	46.92
10	4	10	11.06	39.15
10	4	20	11.17	37.81
10	4	30	11.32	40.41
10	4	40	15.07	37.78
10	4	50	16.46	41.56

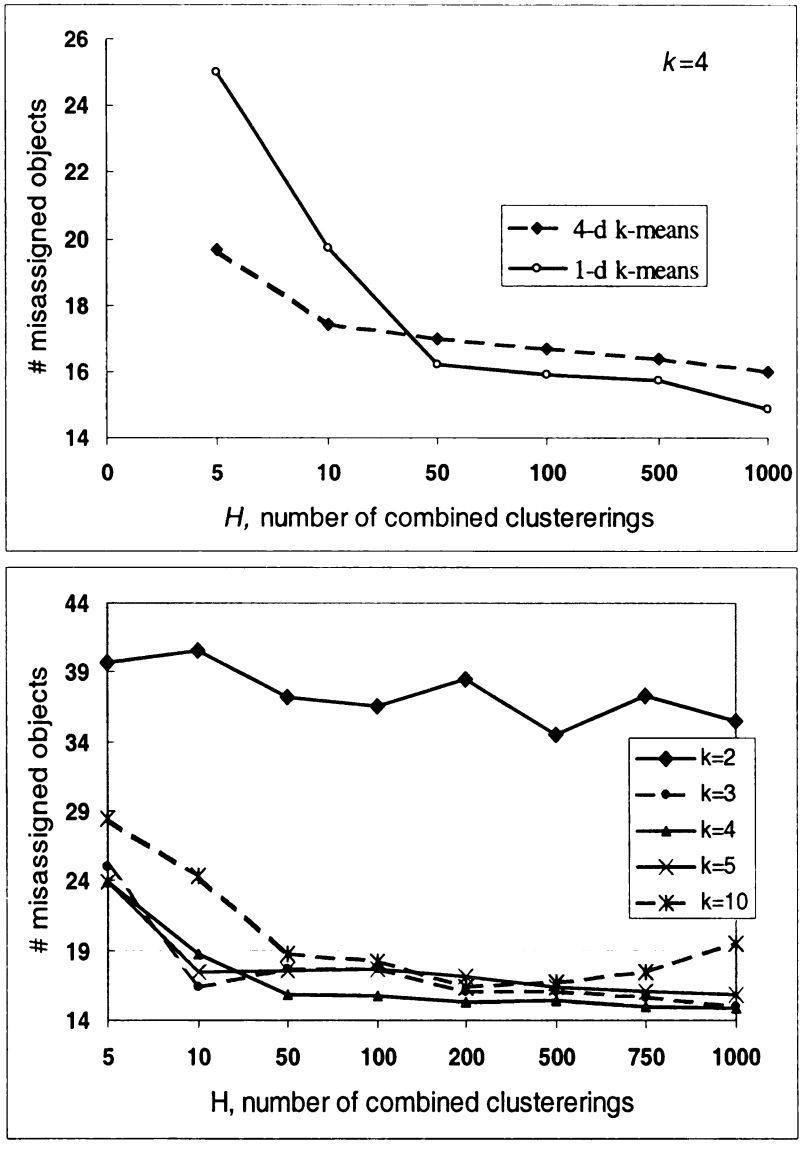


Figure 5.8. Performance of random subspaces algorithm on Iris data. (a) Number of errors by the combination of k-means partitions (k=4) in multidimensional space and projected 1-d subspaces. Average-link consensus function was used. (b) Accuracy of projection algorithm as a function of the number of components and the number of clusters k in each component.

The role of the consensus function is illustrated in Figure 5.9. Three consensus functions are compared on the Iris data set. They all use similarities from the co-

association matrix but cluster the objects using three different criterion functions, namely, single link, average link and complete link. It is clear that the combination using single-link performs significantly worse than the other two consensus functions. This is expected since the three classes in Iris data have hyperellipsoidal shape. More results were obtained on "half-rings" and "2 spirals" data sets in Figure 5.5, which are traditionally difficult for any partitional centroid-based algorithm. Table 5.9 reports the error rates for the "2 spirals" data using seven different consensus functions, different numbers of component partitions H = [5..500] and different numbers of clusters in each component k = 2, 4, 10. Similar results were obtained for the "half-rings" data set under the same experimental conditions and same intermediate values of k. As we see, the single-link consensus function performed the best and was able to identify both the 'half-rings' clusters as well as spirals. In contrast to the results for Iris data, average-link and complete-link consensus were not suitable for these data sets.

Table 5.9. "2 Spirals" data experiments. Average error rate (% over 20 runs) of clustering combination using the random 1-d projections algorithm with different number of components in combination H, different resolutions of components k and seven types of consensus functions.

H, # of components	k, # of cl. in component	Type of Consensus Function: Co-association methods: Single link	Co-association methods: Average link	Co-association methods: Complete link	Hypergraph methods: CSPA	Hypergraph methods: HGPA	Hypergraph methods: MCLA	Median partition: QMI
5	2	47.8	45.0	44.9	41.7	50.0	40.8	40.0
10	2	48.0	44.3	43.1	41.6	50.0	40.4	40.0
50	2	44.2	43.5	41.6	43.1	50.0	41.1	39.3
100	2	46.2	43.9	42.1	46.2	50.0	43.2	42.6
200	2	44.5	42.5	41.9	43.3	50.0	40.1	40.7
500	2	41.6	43.5	39.6	46.4	50.0	44.0	43.3
5	4	48.6	45.9	46.8	43.4	44.9	43.8	44.7
10	4	47.4	47.5	48.7	44.1	43.8	42.6	44.0
50	4	35.2	48.5	46.2	44.9	39.9	42.3	44.2
100	4	29.5	49.0	47.0	44.2	39.2	39.5	42.9
200	4	27.8	49.2	46.0	47.7	38.3	37.2	39.0
500	4	4.4	49.5	44.0	48.1	39.4	45.0	43.4
5	10	48.0	42.6	45.3	42.9	42.4	42.8	45.3
10	10	44.7	44.9	44.7	42.4	42.6	42.8	44.2
50	10	9.4	47.0	44.3	43.5	42.4	42.2	42.8
100	10	0.9	46.8	47.4	41.8	41.1	42.3	44.2
200	10	0.0	47.0	45.8	42.4	38.9	44.5	40.0
500	10	0.0	47.3	43.4	43.3	35.2	44.8	37.4

This observation supports the idea that the accuracy of the consensus functions based on co-association values is sensitive to the choice of data set. In general, one can expect that average-link (single-link) consensus will be appropriate if standard average-link (single-link) agglomerative clustering works well for the data and vice versa. Moreover, none of the three hypergraph consensus functions could find a correct combined partition. This is somewhat surprising given that the hypergraph algorithms performed well on the Iris data. However, the Iris data is far less problematic because one of the clusters is linearly separable, and the other classes are well described as a mixture of two multivariate normal distributions.

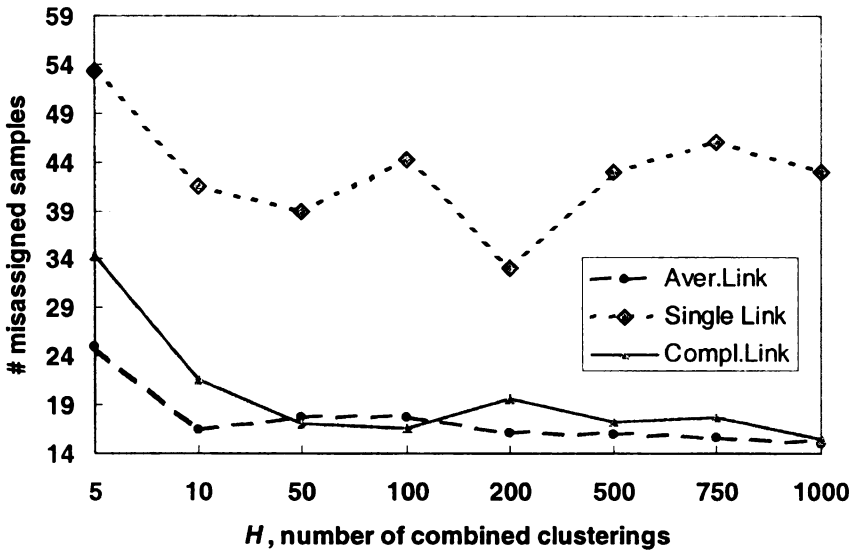


Figure 5.9. Dependence of accuracy of the projection algorithm on the type of consensus function for Iris data set. k=3.

Perfect separation of natural clusters was achieved with a large number of partitions in clustering combination (H > 200) and for values of k > 3 for "half-rings" and "2 spirals". Again, it indicates that for each problem there is a critical value of resolution of component partitions that guarantees good clustering combination. This further supports the work of Fred and Jain [40][41] who showed that a random number of clusters in each partition ensures a greater diversity of components. We see that the minimal required value of resolution for the Iris data is k=3, for "half-rings" it is k=2 and for "2 spirals" it is k=4. In general, the value of k should be larger than the true number of clusters.

The number of partitions affects the relative performance of the consensus functions. With large values of H (>100), co-association consensus becomes stronger, while with small values of H it is preferable to use hypergraph algorithms or the k-means median partition algorithm.

It is interesting to compare the combined clustering accuracy with the accuracy of some of the classical clustering algorithms. For example, for Iris data the EM algorithm

has the best average error rate of 6.17%. In our experiments, the best performers for Iris data were the hypergraph methods, with an error rate as low as 3%, with $H > 200$ and $k > 5$. For the "half-rings" data, the best standard result is 5.25% error by the average-link algorithm, while the combined clustering using the single-link co-association algorithm achieved a 0% error with $H > 200$. Also, for the "2 spirals" data the clustering combination achieves 0% error, the same as by regular single-link clustering. Hence, with an appropriate choice of consensus function, clustering combination outperforms many standard clustering algorithms. However, the choice of good consensus function is similar to the problem of choice of a good conventional clustering algorithm. Perhaps a good alternative to guessing the right consensus function is simply to run all the available consensus functions and then pick the final consensus partition according to the partition utility criterion in Eq. (5.21). We hope to address this in future applications of the method.

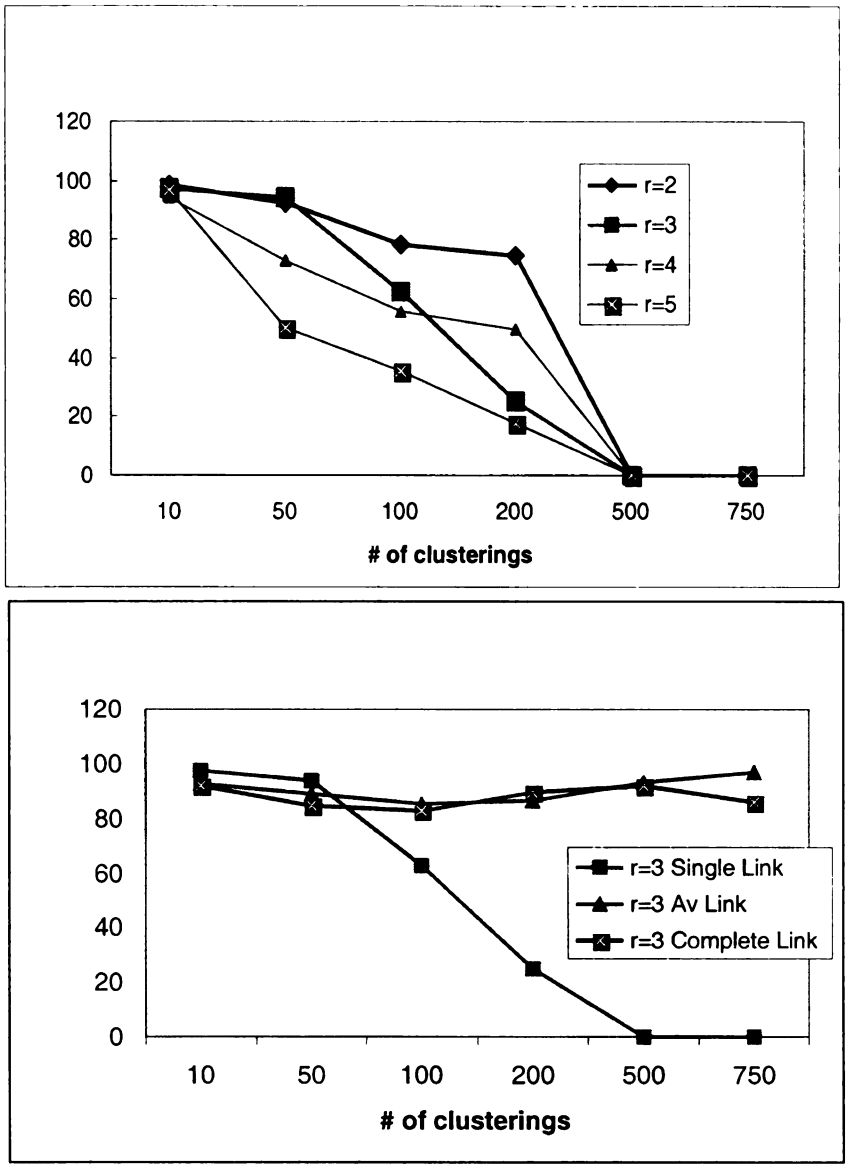


Figure 5.10. Number of misassigned points by random hyperplanes algorithm in clustering of "2 spiral" dataset: (a) for different number of hyperplanes (b) for different consensus functions.

Another set of experiments was performed on the "Galaxy" dataset, which has a significantly larger number of samples, $N = 4192$, and number of features, $d = 14$. The task is to separate patterns of galaxies from stars. We used the most difficult set of "faint" objects from the original data [99]. True labels for the objects were provided manually by experts. Even though computation of component partitions is d times faster

due to projection, overall computational effort can be dominated by the complexity of computing the consensus partition. Quadratic computational complexity effectively prohibits co-association based consensus functions from being used on large data sets, since $O(N^2)$ factor arises when the co-association matrix is built for all the objects. Therefore, for large datasets we do not use the three hierarchical agglomerative methods or the CSPA hypergraph algorithm by Strehl and Ghosh [124]. The k-means algorithm for median partition via QMI is the most attractive in terms of speed, with a complexity $O(kNH)$. In addition, we also used two other hypergraph-based consensus functions, since they worked fast in practice. Table 5.10 reports the achieved error rate using these consensus functions. Also we limited the number of components in combination to H=20 because of the large data size. The results show that the k-means algorithm for median partition has the best performance. HGPA did not work well due to its bias toward balanced cluster sizes, as also happened in the case of the "half-rings" data set. We see that the accuracy improves when the number of partitions and clusters increases.

It is important to note that the average error rate of the standard k-means algorithm for the "Galaxy" data is about 20%, and the best known solution has an error rate of 18.6%. It is quite significant that the k-means median partition algorithm and MCLA obtained a much better partition with an error rate of only around 13% for k>3.

5.6.6 Results of random splitting algorithm

The same set of experiments was performed with clustering combination via splits by random hyperplanes as in Section 5.1. Here we would like to emphasize only the main and the most interesting observations, because the results in many details are close to what has been obtained by using random subspaces. There is a little difference in terms

Table 5.10. Average error rate (%, over 20 runs) of combination clustering using random projections algorithm on "Galaxy/star" data set.

H, # of component	k, # of cl. component	Type of Consensus: Hypergraph methods HGPA	Hypergraph methods MCLA	Median partition QMI
5	2	49.7	20.0	20.4
10	2	49.7	23.5	21.1
20	2	49.7	21.0	18.0
5	3	49.7	22.0	21.7
10	3	49.7	17.7	13.7
20	3	49.7	15.8	13.3
5	4	49.7	19.7	16.7
10	4	49.7	16.9	15.5
20	4	49.7	14.1	13.2
5	5	49.7	22.0	22.5
10	5	49.7	17.7	17.4
20	5	49.6	15.2	12.9

of absolute performance: the random hyperplanes algorithm is slightly better on the "half-rings" data using te single-link consensus function, about the same on "2 spirals", and worse on the Iris data set.

It is important to distinguish the number of clusters k in the component partition and the number of hyperplanes r, because hyperplanes intersect randomly and form varying numbers of clusters. For example, 3 lines can create anywhere between 4 and 7 distinct regions in a plane.

The results for the "2 spirals" data set also demonstrate the convergence of consensus clustering to a perfect solution when H reaches 500 and for the values of r = 2,...,5. See Figure 5.10(a). A larger number of hyperplanes (r=5) improves the convergence. Figure 5.10(b) illustrates that the choice of consensus function is crucial for successful clustering. In the case of the "2-spirals" data, only single-link consensus is able to find correct clusters.

5.7 Adaptive Clustering Ensembles

As we previously discussed, several methods to create partitions for clustering ensembles are known. For example, one can use: (i) different regular clustering algorithms [124], (ii) different initializations, parameter values or built-in randomness of a specific clustering algorithm [38], (iii) weak clustering algorithms, (iv) data resampling [29][35][91]. All these methods generate ensemble partitions independently, in the sense that the probability to obtain the ensemble consisting of H partitions $\{\pi_1, \pi_2,\ldots,\pi_H\}$ of the given data D can be factorized:

$$p(\{\pi_1,\pi_2,...,\pi_H\} \mid D) = \prod_{t=1}^{H} p(\pi_t \mid D)$$

Hence, the increased efficacy of an ensemble is mostly attributed to the number of identically distributed and independent partitions, assuming that a partition of data is treated as a random variable π. Even when the clusterings are generated sequentially, it is traditionally done without considering previously produced clusterings:

$$p(\pi_t \mid \pi_{t-1},\pi_{t-2},...,\pi_1; D) = p(\pi_t \mid D)$$

However, similar to the ensembles of supervised classifiers using boosting algorithms [43], a more accurate consensus clustering can be obtained if contributing partitions take into account the solutions found so far. Unfortunately, it is not possible to mechanically apply the decision fusion algorithms from the supervised (classification) to the unsupervised (clustering) domain. New objective functions for guiding partition generation and the subsequent decision integration process are necessary.

We propose an adaptive approach to partition generation that makes use of clustering history. In clustering, ground truth in the form of class labels is not available. Therefore, we need an alternative measure of performance for an ensemble of partitions. We determine clustering consistency for data points by evaluating a history of cluster assignments for each data point within the generated sequence of partitions. Clustering consistency serves for adapting the data sampling to the current state of an ensemble during partition generation. The goal of adaptation is to improve confidence in cluster assignments by concentrating sampling distribution on problematic regions of the feature space. In other words, by focusing attention on the data points with the least consistent clustering assignments, one can better approximate (indirectly) the inter-cluster boundaries. To achieve this goal, we address the problems related to estimation of clustering consistency and of finding a consensus clustering. Then we evaluate the performance of adaptive clustering ensembles on a number of real-world and artificial data sets in comparison with more conventional clustering ensembles of bootstrap partitions

5.7.1 Adaptive sampling and clustering

While there are many ways to construct diverse data partitions for an ensemble, not all of them easily generalize to adaptive clustering. Our approach extends the studies of ensembles whose partitions are generated via data resampling [29][35]. Intuitively, clustering ensembles generated by other methods can be also boosted. It was shown [91] that small subsamples generally suffice to represent the structure of the entire data set in

the framework of clustering ensembles. Subsamples of small size can reduce computational cost for many exploratory data mining tasks with distributed sources of data [101].

Let D be a data set of N points that is available either as an $N \times d$ pattern matrix in d-dimensional space or an $N \times N$ dissimilarity matrix. Suppose that $X = \{X_1,\ldots,X_H\}$ is a set of H subsamples of size N drawn with replacement from the given data D. A chosen clustering algorithm is run on each of the samples in X that results in H partitions $\Pi=\{\pi_1, \pi_2,\ldots, \pi_H\}$. Each component partition in Π is a set of non-overlapping and exhaustive clusters with $\pi_i = \{ C_1^i , C_2^i ,\ldots, C_{K(i)}^i \}$, $X_i = C_1^i \cup \ldots \cup C_{K(i)}^i$, $\forall \pi_i$ and $K(i)$ is the number of clusters in the i-th partition.

The problem of combining partitions is to find a new partition $\sigma=\{C_1,\ldots,C_M\}$ of the entire data set D given the partitions in Π, such that the data points in a cluster of σ are more similar to each other than to points in different clusters of σ. We assume that the number of clusters M in the consensus clustering is predefined and can be different from the number of clusters k in the ensemble partitions. In order to find this target partition σ, one needs a consensus function utilizing information from the partitions $\{\pi_i\}$. Several known consensus functions [41][124] can be employed to map a given set of partitions $\Pi=\{\pi_1, \pi_2,\ldots, \pi_H\}$ to a target partition σ in our study.

The adaptive partition generation mechanism is aimed at reducing the variance of inter-class decision boundaries. Unlike the regular bootstrap method that draws subsamples uniformly from a given data set, adaptive sampling favors points from regions close to the decision boundaries. At the same time, the points located far from the

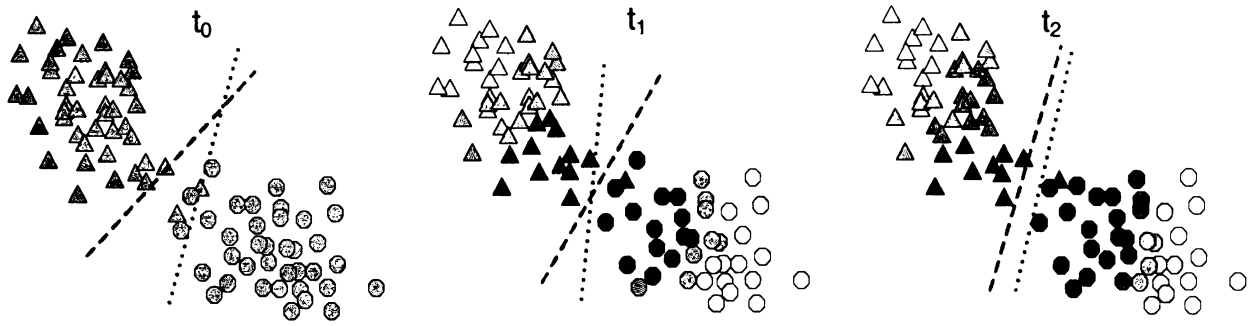


Figure 5.11. Two possible decision boundaries for a 2-cluster data set. Sampling probabilities of data points are indicated by gray level intensity at different iterations ($t_0 < t_1 < t_2$) of the adaptive sampling. True components in the 2-class mixture are shown as circles and triangles.

boundary regions will be sampled less frequently. It is instructive to consider a simple example that shows the difference between ensembles of bootstrap partitions with and without the weighted sampling. Figure 5.11 shows how different decision boundaries can separate two natural classes depending on the sampling probabilities. Here we assume that the *k*-means clustering algorithm is applied to the subsamples. Initially, all the data points have the same weight, namely, the sampling probability $p_i = 1/N$, $i \in [1,\ldots,N]$. Clearly, the main contribution to the clustering error is due to the sampling variation that causes an inaccurate inter-cluster boundary. In contrast, solution variance can be significantly reduced if sampling is increasingly concentrated only on the subset of objects at iterations $t_2 > t_1 > t_0$, as demonstrated in Figure 5.11.

The key issue in the design of the adaptation mechanism is the updating of probabilities. We have to decide how and which data points should be sampled as we collect more and more clusterings in the ensemble. A consensus function based on the co-association values [38] provides the necessary guidelines for adjustments of sampling

probabilities. Remember that the co-association similarity between two data points x and y is defined as the number of clusters shared by these points in the partitions of an ensemble Π:

A consensus clustering can be found by using an agglomerative clustering algorithm (e.g., single linkage) applied to such a co-association matrix constructed from all the points. The quality of the consensus solution depends on the accuracy of similarity values as estimated by the co-association values. The least reliable co-association values come from the points located in the problematic areas of the feature space. Therefore, our adaptive strategy is to increase the sampling probability for such points as we proceed with the generation of different partitions in the ensemble.

The sampling probability can be adjusted not only by analyzing the co-association matrix, which is of quadratic complexity $O(N^2)$, but also by applying the less expensive $O(N+K^3)$ estimation of clustering consistency for the data points. Again, the motivation is that the points with the least stable cluster assignments, namely those that frequently change the cluster they are assigned to, require increased presence in the data

Table 5.11: Consistent re-labeling of 4 partitions

	π_1	π_2	π_3	π_4	π_1'	π_2'	π_3'	π_4'	Consistency
x_1	2	B	X	α	2	1	2	1	0.5
x_2	2	A	X	α	2	2	2	1	0.7
x_3	2	A	Y	β	2	2	1	2	0.7
x_4	2	B	X	β	2	1	2	2	0.7
x_5	1	A	X	β	1	2	2	2	0.7
x_6	2	A	Y	β	2	2	1	2	0.7
x_7	2	B	X	α	2	1	2	1	0.5
x_8	1	B	Y	α	1	1	1	1	1
x_9	1	B	Y	β	1	1	1	2	0.7
x_{10}	1	A	Y	α	1	2	1	1	0.7
x_{11}	2	B	Y	α	2	1	1	1	0.7
x_{12}	1	B	Y	α	1	1	1	1	1

subsamples. In this case, a label correspondence problem must be approximately solved to obtain the same labeling of clusters throughout the ensemble's partitions. By default, the cluster labels in different partitions are arbitrary. To make a correspondence problem more tractable, one needs to re-label each partition in the ensemble using some fixed reference partition. Table 5.11 illustrates how 4 different partitions of twelve points can be re-labeled using the first partition as a reference.

At the (t+1)-st iteration, when some t different clusterings are already included in the ensemble, we use the Hungarian algorithm for the maximum weight bipartite matching problem in order to re-label the (t+1)st partition.

As an outcome of the re-labeling procedure, we can compute the consistency index of clustering for each data point. Clustering consistency index CI at iteration t for a point x is defined as the ratio of the maximal number of times the object is assigned in a certain cluster to the total number of partitions:

$$CI(x) = \frac{1}{H} \max_{L \in \text{ cluster labels}} \left\{ \sum_{i=1}^{H} \delta(\pi_i(x), L) \right\}$$

The values of consistency indices are shown in Table 5.11 after four partitions were generated and re-labeled. We should note that clustering of subsamples of the data set D does not provide the labels for the objects missing (not drawn) in some subsamples.

The clustering consistency index of a point can be directly used to compute its sampling probability. In particular, the probability value is adjusted at each iteration as follows:

$$p_{t+1}(x) = Z(\alpha p_t(x) + CI(x))$$

where α is a discount constant for the current sampling probability and Z is a

normalization factor. The discount factor was set to α=0.3 in our experiments. The proposed clustering ensemble algorithm is summarized in pseudocode below:

Input: D – data set of N points,

H – number of partitions to be combined

M – number of clusters in the consensus partition σ,

K – number of clusters in the partitions of the ensemble,

Γ – chosen consensus function operating on cluster labels

p – sampling probabilities (initialized to $1/N$ for all the points)

Reference Partition $\leftarrow$ k-means(D)

for i=1 to H

Draw a subsample X_i from D using sampling probabilities **p**

Cluster the sample X_i: $\pi(i) \leftarrow k$-means(X_i)

Re-label partition $\pi(i)$ using the reference partition

Compute the consistency indices for the data points in D

Adjust the sampling probabilities **p**

end

Apply consensus function Γ to ensemble Π to find the partition σ

Validate the target partition σ (optional)

return σ *// consensus partition*

5.7.2 Empirical Study of Adaptive Ensembles

The experiments were conducted on artificial and real-world data sets ("Galaxy", "half-rings", "wine", "3-gaussian", "Iris", "LON"), with known cluster labels, to validate the accuracy of consensus partition. A comparison of the proposed adaptive and previous non-adaptive ensembles is the primary goal of the experiments. We evaluated the

performance of the clustering ensemble algorithms by matching the detected and the known partitions of the datasets. The best possible matching of clusters provides a measure of performance expressed as the misassignment rate. To determine the clustering error, one needs to solve the correspondence problem between the labels of known and derived clusters. Again, the Hungarian algorithm was used for this purpose. The *k*-means algorithm was used to generate the partitions of samples of size N drawn with replacement, similar to bootstrap, albeit with dynamic sampling probability. Each experiment was repeated 20 times and average values of error (misassignment) rate are shown in Figure 5.12.

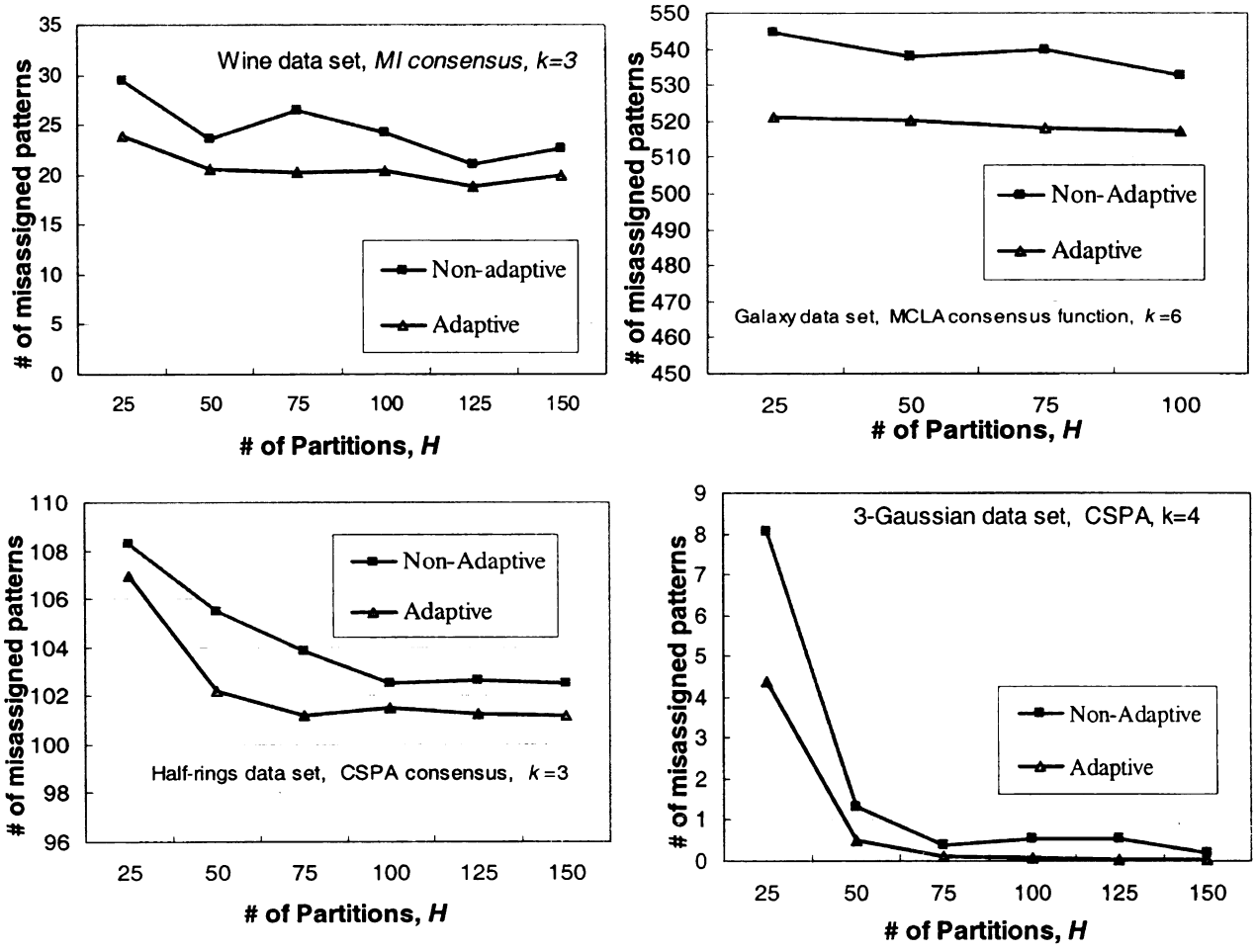


Figure 5.12. Clustering accuracy for ensembles with adaptive and non-adaptive sampling mechanisms as a function of ensemble size for some data sets and selected consensus functions.

Consensus clustering was obtained by four different consensus functions: hypergraph-based MCLA and CSPA algorithms [124], quadratic mutual information and EM algorithm based on the mixture model. The main observation is that adaptive ensembles slightly outperform the regular sampling schemes on most benchmarks. Typically, the clustering error decreased by 1-5%. However, accuracy improvement depends on the number of clusters in the ensemble partitions (K). Generally, the adaptive ensembles were superior for values of K larger than the target number of clusters, M, by 1

or 2. With either too small or too large a value of K, the performance of adaptive ensembles was less robust and occasionally worse than corresponding non-adaptive algorithms. A simple inspection of probability values always confirmed the expectation that points with large clustering uncertainty are drawn more frequently.

Most significant progress was detected when the combination consisted of 25-75 partitions. Large numbers of partitions (H>75) almost never lead to further improvement in clustering accuracy. Moreover, for H>125 we often observed increased error rates (except for the hypergraph-based consensus functions), due to the increase in complexity of the consensus model and in the number of model parameters requiring estimation.

To summarize, we have extended the clustering ensemble framework by an adaptive data sampling mechanism for generation of partitions. We dynamically update sampling probability to focus on more uncertain and problematic points by on-the-fly computation of clustering consistency. Empirical results demonstrate improved clustering accuracy and faster convergence as a function of the number of partitions in the ensemble.

5.8 Conclusions

We advanced previous research on clustering ensembles in several respects. First, we have introduced a unified representation for multiple clusterings and formulated the corresponding categorical clustering problem. Second, we have proposed a solution to the problem of clustering combination. A consensus clustering is derived from a solution of the maximum likelihood problem for a finite mixture model of the ensemble of partitions. An ensemble is modeled as a mixture of multivariate multinomial distributions in the space of cluster labels. The maximum likelihood problem is effectively solved using the EM algorithm. The EM-based consensus function is also capable of dealing with incomplete contributing partitions. Third, it is shown that another consensus function can be related to the classical intra-class variance criterion using the generalized mutual information definition. A consensus solution based on quadratic mutual information can be efficiently found by the k-means algorithm in the space of specially transformed cluster labels. Experimental results indicate good performance of the approach for several datasets and favorable comparison with other consensus functions. Among the advantages of these algorithms are their low computational complexity and well-grounded statistical model.

We have also considered combining weak clustering algorithms that use data projections and random data splits. A simple explanatory model is offered for the behavior of combination of such weak components. We have analyzed combination accuracy as a function of parameters that control the power and resolution of component

partitions as well as the learning dynamics vs. the number of clusterings involved. Empirical study compared effectiveness of several consensus functions applied to weak partitions.

It is interesting to continue the study of clustering combination in several directions: 1) design of effective search heuristics for consensus functions, 2) a more precise and quantifiable notion of weak clustering, and 3) an improved understanding of effect of component resolution on overall performance. This research can be extended in order to take into account non-independence of partitions in the ensemble. The consensus function presented here is equivalent to a certain kind of Latent Class Analysis, which offers established statistical approaches to measure and use dependencies (at least pair-wise) between variables. It is also interesting to consider a combination of partitions of different quality. In this case one needs to develop a consensus function that weights the contributions of different partitions proportionally to their strength. We hope to address these issues in our future work.

Chapter 6

Summary

This dissertation addresses three information-processing problems, which are central to modern data mining, and advances their solution by using evolutionary computation and ensemble approaches.

Feature Extraction. The goal is to find the best subset of original variables or their transformation relevant to a computational model. In general, new variables can be synthesized from existing data features. Generalization or predictive classification accuracy of the model is the main evaluation criterion in feature selection and extraction.

Our contribution to this problem is evolutionary feature extraction and dimensionality reduction based on genetic search of optimal combinations of feature values. Accuracy-driven construction of new features uses intelligent value clustering. We offer feature extraction methods for improved pattern classification using genetic algorithms. New features are synthesized by merging the values of original variables during the search process. In this study, we introduce a different GA approach to dimensionality reduction that does not explicitly operate with whole features, but instead operates on individual values assumed by the feature variables. Features are extracted by clustering several "old" values into a new meta-value which substitutes for the old values in the feature vector. Therefore, new features are created by clustering the values of variables. A GA is

used as a search engine for this value clustering. If features assume nominal values then clustering could be viewed as a grouping problem. Scalability and redundancy are two characteristic problems of grouping algorithms. Here we present a grouping GA with an improved graph-based encoding. The proposed cluster representation has lower redundancy while preserving the simplicity of decoding and evaluation. Compact solution representation with minimal redundancy of search space is designed for a wide class of grouping problems, including clustering of variable values. This encoding is not limited to value clustering and therefore applicable to other grouping tasks. The genetic search of (sub-) optimal combinations of values is performed in graph-based encoding of candidate solutions. Genetic value clustering is applied to text categorization, DNA-based assignments of individuals in population genetics and parametric learning of Bayesian network classifiers. It is shown that such feature extraction results in better predictive accuracy of classification decisions.

Data model identification. Prediction of output variables is based on learning of an unknown function for a given training sample and a set of elementary functional components. We propose fast genetic programming for data-driven induction of predictive models with real-valued input-output mappings. We develop genetic programming algorithms for modeling of input-output relations of continuous variables that incorporates dynamical fitting of free parameters of evolved models. Traditional genetic programming is extended by gradient descent optimization of leaf coefficients of tree-like programs during the evolutionary search that is made possible using algorithmic

differentiation. Experimental results show significant improvement in both computational requirements and modeling accuracy for a set of symbolic regression problems.

Data clustering. Partitioning of a data set must be done into a number of meaningful groups of data points. Clustering criteria use available data features and a variety of prior knowledge about data distribution properties. We propose generic and robust methods to combine multiple clusterings of data including the use of novel fast algorithms for consensus clustering as well as inexpensive and weak clustering components.

Ensembles of partitions of data sets are studied in two respects: combination of multiple clusterings and generation of clusterings for an ensemble. We develop two efficient consensus functions for finding a combined partition of good quality. First consensus function uses information-theoretic principle based on maximal generalized mutual information. Second function finds a consensus clustering by estimation of a probabilistic mixture model from the observed ensemble. It is demonstrated that ensemble's partitions can be generated by the weak clustering algorithms, in particular, by clusterings in random low-dimensional subspaces of original feature space. Experiments indicate that ensemble of weak partitions can be more accurate than a single sophisticated clustering algorithm. Finally, we consider how partition generation process can adapt to provide better decisions for the patterns near the boundaries between the data clusters.

The main results of this dissertation were published in [52][53][66][91] [128][129][130][131][132].

Appendix

EM Algorithm for Consensus Solution

Here we provide detailed derivation of the EM algorithm for estimation of parameters of the mixture model proposed in Section 5.3.

The distribution of hidden variable **z** should be consistent with the observed values ***Y***:

$$\log P(\mathbf{Y} \mid \Theta) = \log \sum_{z \in \mathbf{Z}} P(\mathbf{Y}, \mathbf{z} \mid \Theta) . \tag{A1}$$

If the value of $\mathbf{z}_i$ is known then one could immediately tell which of the M mixture components was used to generate the point $\mathbf{y}_i$. It means that with each observed data point $\mathbf{y}_i$, we associate a hidden vector variable $\mathbf{z}_i = \{z_{i1},\ldots,z_{iM}\}$ such that $z_{im} = 1$ if $\mathbf{y}_i$ belongs to the m-th component and $z_{im} = 0$, otherwise. It is convenient to write the complete data likelihood as:

$$\begin{aligned} \log L(\Theta \mid \mathbf{Y}, \mathbf{Z}) &= \log \prod_{i=1}^{N} P(\mathbf{y}_i, \mathbf{z}_i \mid \Theta) = \log \prod_{i=1}^{N} \prod_{m=1}^{M} \left(\alpha_m P_m(\mathbf{y}_i \mid \boldsymbol{\theta}_m)\right)^{z_{im}} \\ &= \sum_{i=1}^{N} \sum_{m=1}^{M} z_{im} \log \alpha_m P_m(\mathbf{y}_i \mid \boldsymbol{\theta}_m) . \end{aligned} \tag{A2}$$

According to the general EM approach, we have to define an auxiliary function $Q(\Theta;\Theta')$ that serves as a lower bound on the observed data likelihood in Eq. (5.3):

$$Q(\Theta;\Theta') = \sum_{z} \log\left(P(\mathbf{Y}, \mathbf{z} \mid \Theta)\right) p(\mathbf{z} \mid \mathbf{Y}, \Theta'). \tag{A3}$$

Classical convergence analysis of EM algorithm [25][28] establishes that the maximization of the function $Q(\Theta;\Theta')$ with respect to Θ is equivalent to increasing the observed likelihood function in Eq. (5.3). Evaluation of $Q(\Theta;\Theta')$ is the first step of the EM algorithm. Substitution of Eq. (A2) in the definition of Q function gives:

$$\begin{aligned} Q(\Theta;\Theta') &= \sum_{z}\sum_{i=1}^{N}\sum_{m=1}^{M} z_{im}\log\alpha_m P_m(\mathbf{y}_i|\boldsymbol{\theta}_m)p(\mathbf{z}\,|\,\mathbf{Y},\Theta') \\ &= \sum_{i=1}^{N}\sum_{m=1}^{M} E[z_{im}]\,\log\alpha_m P_m(\mathbf{y}_i|\boldsymbol{\theta}_m). \end{aligned} \tag{A4}$$

The last expression depends on the previous estimate of parameters Θ' only via the expected values of the hidden variables $E[z_{im}]$, which are defined as:

$$E[z_{im}] = \sum_{z} z_{im}\; p(\mathbf{z}\,|\,\mathbf{Y},\Theta') = \frac{\alpha'_m P_m(\mathbf{y}_i|\boldsymbol{\theta}'_m)}{\sum_{n=1}^{M}\alpha'_n P_n(\mathbf{y}_i|\boldsymbol{\theta}'_n)}. \tag{A5}$$

Here, the current guess about the parameters $\Theta' = \{\alpha'_1,\ldots,\alpha'_M,\boldsymbol{\theta}'_1,\ldots,\boldsymbol{\theta}'_M\}$ is used to compute the expectations. Taking into account the concrete form of the component-conditional densities $P_m(\mathbf{y}_i|\boldsymbol{\theta}_m)$ from Eqs. (5.5) and (5.6), we obtain the E-step of the algorithm:

$$E[z_{im}] = \frac{\alpha'_m\prod_{j=1}^{H}\prod_{k=1}^{K(j)}\left(\vartheta'_{jm}(k)\right)^{\delta(y_{ij},k)}}{\sum_{n=1}^{M}\alpha'_n\prod_{j=1}^{H}\prod_{k=1}^{K(j)}\left(\vartheta'_{jn}(k)\right)^{\delta(y_{ij},k)}}. \tag{A6}$$

The M-step consists of maximizing the Q function in Eq. (A4) by the parameters Θ, given the expected values of the hidden variables $E[z_{im}]$ from E-step in Eq. (A6):

$$\Theta^* = \arg\max_{\Theta} Q(\Theta;\Theta') = \arg\max_{\{\alpha_m,\vartheta_{jm}\}} \sum_{i=1}^{N}\sum_{m=1}^{M}\left(E[z_{im}]\log\alpha_m + E[z_{im}]\log P_m(\mathbf{y}_i|\boldsymbol{\theta}_m)\right). \quad \text{(A7)}$$

The two terms on the right hand side can be optimized independently. The expression for the coefficients α_m is easily found using a Lagrange multiplier along with the constraint $\Sigma_m \alpha_m = 1$:

$$\frac{\partial Q(\Theta;\Theta')}{\partial \alpha_m} = \frac{\partial}{\partial \alpha_m}\left(\sum_{i=1}^{N}\sum_{m=1}^{M} E[z_{im}]\log\alpha_m + \lambda\left(\sum_{m=1}^{M}\alpha_m - 1\right)\right) = 0 \quad \text{(A8)}$$

$$\alpha_m = \frac{\sum_{i=1}^{N} E[z_{im}]}{\sum_{i=1}^{N}\sum_{m=1}^{M} E[z_{im}]}. \quad \text{(A9)}$$

Similarly, obtaining the optimizing values of $\vartheta_{jm}(k)$, is facilitated by the independence assumption for the component-conditional densities of variables y_{ij} as described by Eq. (5.3). Again, natural constraints $\Sigma_k \vartheta_{jm}(k) = 1$ and Lagrange multipliers λ_{jm} are utilized:

$$\frac{\partial Q(\Theta;\Theta')}{\partial \vartheta_{jm}(k)} = \frac{\partial}{\partial \vartheta_{jm}(k)}\left(\sum_{i=1}^{N}\sum_{m=1}^{M} E[z_{im}]\log P_m(\mathbf{y}_i|\boldsymbol{\theta}_m) + \lambda_{jm}\left(\sum_{k=1}^{K(j)}\vartheta_{jm}(k) - 1\right)\right) = 0 \quad \text{(A10)}$$

$$\vartheta_{jm}(k) = \frac{\sum_{i=1}^{N}\delta(y_{ij},k)E[z_{im}]}{\sum_{i=1}^{N}\sum_{k=1}^{K(j)}\delta(y_{ij},k)E[z_{im}]}. \quad \text{(A11)}$$

To summarize, the EM algorithm starts with an initial guess for the values of the parameters $\{\alpha_1',\ldots,\alpha_M',\boldsymbol{\theta}_1',\ldots,\boldsymbol{\theta}_M'\}$ of the mixture density. After this, E and M steps are repeated until a chosen convergence criterion is satisfied. The *E*-step computes the

expected values of the hidden variables $E[z_{im}]$ according to Eq. (A6). The M-step maximizes the likelihood by computing new best parameters estimates according to Eqs. (A9, A11). The convergence criteria can be based on the increase in the amount of the likelihood function between two consequent M-steps or on the stability of the assignment of points from $\boldsymbol{Y}$, or equivalently from $\boldsymbol{X}$. In fact, it is the stability criterion that is more relevant to the clustering task. Intuitively, each E-step is essentially equivalent to the naïve Bayes computation procedure. However, the M-step uses "soft" real-valued cluster memberships to determine pattern contributions to the component-conditional densities, in contrast to conventional supervised maximum likelihood estimation.

Bibliography

[1] E. E. Abola, F. C. Bernstein, S. H. Bryant, T. F. Koetzle, and J. Weng, Protein Data Bank, In Crystallographic Databases–Information Content, Software Systems, Scientific Applications (F. Allen et al. eds), Data Commission of the Intl. Union of Crystallography, Bonn/Cambridge/Chester, 107–132, 1987.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, In Proceedings of ACM-SIGMOD 1998 Int. Conf. on Management of Data: 94-105, 1998.

[3] H. Almuallim, and T.G. Dietterich, Learning with many irrelevant features. In Proceedings of the Ninth National Conference on Artificial Intelligence, 547-552. Menlo Park, CA: AAAI Press, 1991.

[4] H. Almuallin, and T.G. Dietterich, Efficient algorithms for identifying relevant features. Proceedings of the Ninth Canadian Conference on Artificial Intelligence (pp. 38-45). Vancouver, BC:Morgan Kaufmann, 1992.

[5] P.G. Angeline, G. Saunders, and J. Pollak, An Evolutionary Algorithm That Constructs Recurrent Neural Networks, IEEE Trans. On Neural Networks, Vol. 5, No. 1, 54-65, 1994.

[6] J.-P. Barthelemy and B. Leclerc, The median procedure for partition, In Partitioning Data Sets, I.J. Cox et al eds., AMS DIMACS Series in Discrete Mathematics, 19: 3-34, 1995.

[7] D. J. Bartholomew and M. Knott, Latent variable models and factor analysis, 2nd ed, Kendall's Library of Statistics 7. London: Arnold, 1999.

[8] R.K. Belew, J. McInerney, and N.N. Schraudolph, Evolving networks: Using the Genetic Algorithm with connectionist learning. In Proceedings of the Second Artificial Life Conference, 511-547, Addison-Wesley, 1991.

[9] A. Ben-Hur, D. Horn, H.T. Siegelmann, and V.Vapnik, Support vector clustering, Journal of Machine Learning Research, 2: 125 - 137, 2001.

[10] L. Breiman, Bagging Predictors, Machine Learning, Vol. 24 (2), 123 - 140, 1996.

[11] L. Breiman, Arcing classifiers, The Annals of Statistics, 26 (3): 801 - 849, 1998.

[12] L. Breiman, J. Friedman, R. Olshen, and C. Stone, Classification and Regression Trees. Wadsworth International, California, 1984.

[13] B. Bollobas, Random Graphs, Academic Press, London, 1985.

[14] S.P. Brumby, J. Theiler, S.J. Perkins, N.R. Harvey, J.J. Szymanski, J.J. Bloch, and M. Mitchell, Investigation of Feature Extraction by a Genetic Algorithm, Proc. SPIE 3812, 24-31, 1999.

[15] T. Cai, N. Cercone, and J. Han, Attribute-oriented induction in relational databases. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, Knowledge Discovery in Databases, 213-228. AAAI Press/The MIT Press, 1991.

[16] C. Chang, Dynamic Programming as applied to Feature Subset Selection in a Pattern Recognition System, IEEE Transactions on Systems, Man and Cybernetics, No. 3, 166 - 171, 1973.

[17] G. Chartrand, and O. R. Oellermann, Applied and Algorithmic Graph Theory. McGraw-Hill, Inc., New York, 1993.

[18] C. K. Chow and C. N. Liu, Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory, 14: 462-467, 1968.

[19] D. Cristofor and D.A. Simovici, An information-theoretical approach to clustering categorical databases using genetic algorithms. In Proceedings of 2nd SIAM ICDM Workshop on clustering high dimensional data, 2002.

[20] T. M. Cover, The Best Two Independent Measurements are Not the Two best, IEEE Transactions on Systems, Man, and Cybernetics, pp. 116-117, 1974.

[21] T. M. Cover and J. M. Van Campenhout, On the possible orderings in the measurement selection problem. IEEE Transactions on Systems Man and Cybernetics, SMC-7: 657-661, 1977.

[22] R. Cucchiara, Genetic algorithms for clustering in machine vision. Machine Vision and Applications, 11: 1 – 6, 1998.

[23] A.C. Davison and D.V. Hinkley, Bootstrap Methods and their Application, Cambridge University Press, second edition, 1997.

[24] E. Dimitriadou, A. Weingessel and K. Hornik, Voting-merging: An ensemble method for clustering, In Proc. Int. Conf. on Artificial Neural Networks, Vienna, 217-224, 2001.

[25] A. P. Dempster, N. M. Laird, and D. B. Rubin, Maximum Likelihood From Incomplete Data Via the EM Algorithm, Journal of the Royal Statistical Society B, 39: 1-22, 1997.

[26] P. A. Devijver and J. Kittler, Pattern Recognition: A Statistical Approach, Prentice-Hall International, 1982.

[27] P. Domingos and M. Pazzani, On the optimality of the simple Bayesian classifier under zero-one loss, Machine Learning, 29: 103–130, 1997.

[28] R.O. Duda, P.E. Hart, and D.G. Stork, Pattern Classification, 2nd Ed, John Wiley & Sons Inc., 2001.

[29] S. Dudoit and J. Fridlyand, Bagging to improve the accuracy of a clustering procedure, Bioinformatics, 19 (9): 1090-1099, 2003.

[30] A. Esparcia-Alcazar and K. Sharman, Learning schemes for genetic programming, In Proceedings Late Breaking Papers at the 1997 Genetic Programming Conference, 57- 65, Stanford University, CA, 1997

[31] E. Falkenauer, A New Representation and Operators for GAs Applied to Grouping Problems, Evolutionary Computation, Vol.2 (2), 123-144, 1994.

[32] E. Falkenauer, Genetic Algorithms and Grouping Problems, John Wiley & Son Ltd., 1998.

[33] S.E. Fahlman and C. Lebiere, The cascade-correlation learning architecture. In D. Touretzky, editor, Advances in Neural Information Processing Systems, Vol. 2, pages 524-532. Morgan Kaufmann, 1990.

[34] X. Z. Fern and C. E. Brodley, Random Projection for High Dimensional Data Clustering: A Cluster Ensemble Approach, Proc. of 20th Intl. Conf. on Machine learning, 2003.

[35] B. Fischer, J.M. Buhmann, Path-Based Clustering for Grouping of Smooth Curves and Texture Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, 25 (4): 513-518, 2003.

[36] D. H. Fisher, Knowledge acquisition via incremental conceptual clustering, Machine Learning, 2: 139-172, 1987.

[37] M. Figueiredo and A.K. Jain, Unsupervised learning of finite mixture models, IEEE Transaction on Pattern Analysis and Machine Intelligence 24: 381-396, 2002.

[38] A.L.N. Fred, Finding Consistent Clusters in Data Partitions, In Proc. 3d Int. Workshop on Multiple Classifier Systems. Eds. F. Roli, J. Kittler, LNCS 2364: 309-318, 2001.

[39] A.L.N. Fred and A.K. Jain, Data Clustering using Evidence Accumulation, In Proc. of the 16th Intl. Conference on Pattern Recognition, ICPR 2002 ,Quebec City: 276 – 280, 2002.

[40] A.L.N. Fred and A.K Jain, Evidence Accumulation Clustering based on the K-means algorithm, In T.Caelli et al., eds, Structural, Syntactic, and Statistical Pattern Recognition, LNCS 2396, Springer-Verlag, 442-451, 2002.

[41] A. Fred and A. K. Jain. Robust data clustering, In Proc. of IEEE Computer Society Conf, on Computer Vision and Pattern Recognition, CVPR 2003, Wisconsin, USA, June 2003.

[42] A.A. Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag, 2002.

[43] Y. Freund and R.E. Schapire, Experiments with a New Boosting Algorithm, in Proc. of the Thirteenth Intl. Conference on Machine Learning, Morgan Kaufmann, 148-156, 1996.

[44] N. Friedman, D. Geiger, and N. Lotner, Likelihood Computation with Value Abstraction, In Proc. Sixteenth Conf. on Uncertainty in Artificial Intelligence (UAI), 2000.

[45] N. Friedman, O. Mosenzon, N. Slonim, and N. Tishby, Multivariate Information Bottleneck, in Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence, 2001.

[46] W. Gablentz, M. Köppen, and E. Dimitriadou, Robust Clustering by Evolutionary Computation, In Proc. 5th Online World Conf. on Soft Computing in Industrial Applications (WSC5), 2000.

[47] Z. Ghahramani and M. Jordan, Supervised learning from incomplete data via an EM approach, In Proceedings of Advances in Neural Information Processing Systems (NIPS 6): 120-127, 1993.

[48] M.A. Gluck and J.E. Corter, Information, uncertainty, and the utility of categories. In Proc. of the Seventh Annual Conference of the Cognitive Science Society, Hillsdale, NJ: Lawrence Erlbaum, 283-287, 1985.

[49] L.A. Goodman and W.H. Kruskal, Measures of association for cross-classifications. Journal of Am. Stat. Assoc.; 49: 732 - 764, 1954.

[50] J. Ghosh, A. Strehl, and S. Merugu, A consensus framework for integrating distributed clusterings under limited knowledge sharing. In Proc. NSF Workshop on Next Generation Data Mining, pages 99-108, 2002.

[51] A. Griewank, Evaluating derivatives: Principles and techniques of algorithmic differentiation, SIAM, Philadelphia, 2000.

[52] B. Guinand, K.T. Scribner, A. Topchy, K.S. Page, W. Punch & M.K. Burnham-Curtis, Sampling issues affecting accuracy of likelihood-based classification using genetical data, Environmental Biology of Fishes, vol. 69: 245–259, 2004.

[53] B. Guinand, A. Topchy, K. S. Page, M. K. Burnham-Curtis, W. F. Punch, and K.T. Scribner, Comparisons of likelihood and machine learning methods of individual classification. Journal of Heredity 93(4), 260-269, 2002.

[54] M. Halkidi,Y. Batistakis,and M. Vazirgiannis, Clustering algorithms and validity measures. In Proceedings of the SSDBM conference, Virginia, USA, July 2001.

[55] H. J. Hamilton, R. J. Hilderman, and N. Cercone, Attribute-oriented induction using domain generalization graphs. In Proceedings of the Eighth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'96), 246-253, Toulouse, France, 1996.

[56] J. Havrda and F. Charvát, Quantification Method of Classification Processes: Concept of Structrual a-Entropy, Kybernetika, 3: 30-35, 1967.

[57] A. Hinneburg, D. A. Keim, and M. Wawryniuk, Using Projections to visually cluster high-dimensional Data, Computing in Science and Engineering, 5 (2): 12-25, 2003.

[58] G. E. Hinton and S. J Nowlan, How learning can guide evolution, Complex Systems, 1, 495-502, 1987.

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

[60] G. E. Hughes, On the mean accuracy of statistical pattern recognition. IEEE Transactions on Information Theory, IT-14 (1): 55 - 63, 1968.

[61] M. Hutter, Distribution of Mutual Information, in Advances in Neural Information Processing Systems, 14, 399-406, 2002.

[62] H. Iba and N. Nikolaev, Genetic Programming Polynomial Models of Financial Data Series," in Proceedings of the Conference on Evolutionary Computation, CEC-2000, IEEE Press, pp. 1459-1466, 2000.

[63] A.K. Jain and R.C. Dubes, Algorithms for Clustering Data, Prentice-Hall, New Jersey, 1988.

[64] A.K. Jain, R.P. Duin and J. Mao, Statistical pattern recognition: a review, IEEE Transactions on Pattern Analysis and Machine Intelligence 22: 4 - 37, 2000.

[65] A.K. Jain, M. N. Murty, and P. Flynn, Data clustering: A review, ACM Computing Surveys, 31(3): 264 – 323, 1999.

[66] A.K. Jain, A. Topchy, M. Law, Landscape of Clustering Algorithms, In Proc. Intl. Conf on Pattern Recognition, ICPR 2004, Cambridge, UK, Aug 2004.

[67] E. Johnson and H. Kargupta, Collective, hierarchical clustering from distributed, heterogeneous data, In Large-Scale Parallel KDD Systems. Eds. Zaki M. and Ho C., Volume 1759 of LNCS, Springer-Verlag, 221–244, 1999.

[68] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, Multilevel Hypergraph Partitioning: Applications in VLSI Design, In Proc. ACM/IEEE Design Automation Conf., 526-529, 1997.

[69] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal of Scientific Computing, 20(1): 359 - 392, 1998.

[70] P. Kellam, X. Liu, N.J. Martin, C. Orengo, S. Swift, and A.Tucker, Comparing, contrasting and combining clusters in viral gene expression data, Proceedings of 6th Workshop on Intelligent Data Analysis in Medicine and Pharmocology, 56-62, 2001.

[71] E.M. Kleinberg, Stochastic Discrimination, Annals of Mathematics and Artificial Intelligence, 1: 207-239, 1990.

[72] J. Kleinberg, An Impossibility Theorem for Clustering, In Proceedings of Advances in Neural Information Processing Systems (NIPS 2002) 15, 2002.

[73] J. Kittler, Feature set search algorithms, in Pattern Recognition and Signal Processing, ed., C.H. Chen, pp. 41-60, Sitho and Noordho, 1978.

[74] R. Kohavi, G. John, Wrappers for Feature Subset Selection, Artificial Intelligence journal 97 (1-2), 273-324, 1997.

[75] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: The MIT Press, 1992.

[76] J.R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs. Cambridge, MA: The MIT Press, 1994.

[77] J.R. Koza, Bennett III, Forrest H, Andre, David, and Keane, Martin A.. Genetic Programming III: Darwinian Invention and Problem Solving. San Francisco, CA: Morgan Kaufmann Publishers, 1999.

[78] J.R. Koza, F. H Bennett III, D. Andre, M.A. Keane, and F. Dunlap, Automated synthesis of analog electrical circuits by means of genetic programming, IEEE Transactions on Evolutionary Computation, 1(2), 109-128, 1997.

[79] P. Langley, W. Iba, and K. Thompson, An analysis of Bayesian classifiers, In Proc. of the Tenth National Conf. on Artificial Intelligence, San Jose, CA, AAAI Press, 399–406, 1992.

[80] W. B. Langdon, Data Structures and Genetic Programming: Genetic Programming + Data Structures = Automatic Programming, Kluwer, Boston, 1998.

[81] O. Larsen, A.A. Freitas and J.C. Nievola, Constructing X-of-N attributes with a genetic algorithm. Proc. 4th Int. Conf. on Recent Advances in Soft Computing, 326-331, 2002.

[82] E. L. Lawler and D. E. Wood, Branch and Bound Methods: A Survey. Operations Research 14, 699-719, 1966.

[83] G. von Laszewsski and H. Muhlenbein, Partitioning a graph with a Parallel Genetic Algorithm. In Proceedings of Parallel Problem Solving from Nature, PPSN 1, H.-P.Schwefel and R.Manner (Eds.), Springer-Verlag, Berlin, Germany, 165-169, 1990.

[84] F. Leisch, Bagged clustering, Working Papers SFB "Adaptive Information Systems and Modeling in Economics and Management Science", no.51, Aug. 1999, Institut für Information, Abt. Produktionsmanagement, Wien, Wirtschaftsuniv, 1999.

[85] C. Lin and J.Wu, Automatic facial feature extraction by genetic algorithms, IEEE Trans. Image processing, vol. 8, 6, 834-845, 1999.

[86] H. Liu, H. Motoda, and M. Dash, A monotonic measure for optimal feature selection. In Proceedings of European Conference on Machine Learning, pages 101 - 106, 1998.

[87] J. Mao, K. Mohiuddin, and A. K. Jain, Parsimonious network design and feature selection through node pruning. In Proceedings of 12th ICPR, Jerusalem, 622 - 624, 1994.

[88] M. Martin-Bautista and M.-A. Vila, A survey of genetic feature selection in mining issues, in Proceedings of the Congress on Evolutionary Computation (CEC 99), 13-23, 1999.

[89] M. Merzbacher and W. W. Chu, Pattern-based clustering for database attribute values. In Proceedings of AAAI Workshop on Knowledge Discovery in Databases, Wash., D.C., 1993.

[90] R.S. Michalski and R.E Stepp, Automated construction of classifications: Conceptual clustering versus numerical taxonomy, IEEE Trans. Pattern Analysis and Machine Intelligence, 5, 396-410, 1983.

[91] B. Minaei, A. Topchy, and W. Punch, Ensembles of Partitions via Data Resampling, Proc. Intl. Conf. Information Technology, ITCC 2004, Las Vegas, 2004.

[92] J. Minker, G.A. Wilson, B.H. Zimmerman, An evaluation of query expansion by the addition of clustered terms for a document retrieval system, Information Storage and Retrieval 8(6), 329-48, 1972.

[93] B. Mirkin, Reinterpreting the Category Utility Function, Machine Learning, 45(2): 219-228, 2001.

[94] S. Monti, P. Tamayo, J. Mesirov, and T. Golub, Consensus Clustering: A Resamlping Based Method for Class Discovery and Visualization of Gene Expression Microarray Data, Journal on Machine Learning, 52 (1-2), 2003

[95] C.A. Murthy and N. Chowdhury, In search of optimal clusters using genetic algorithms. Pattern Recognition Letters, 17:825–832, 1996.

[96] P. M. Narendra and K. Fukunaga, A branch and bound algorithm for feature subset selection. IEEE Transactions on Computers, C-26: 917–922, 1977.

[97] S. Nolfi, J. Elman, and D. Parisi, Learning and evolution in neural networks, Adaptive Behavior, 3(1), 5 - 28, 1994.

[98] J.R. O'Connell, D.E. Weeks, The VITESSE algorithm for rapid exact multilocus linkage analysis via genotype set-recoding and fuzzy inheritance, Nature Genetics 11, 402-408, 1995.

[99] S.C. Odewahn, E.B. Stockwell, R.L. Pennington, R.M. Humphreys, and W.A. Zumach, Automated Star/Galaxy Discrimination with Neural Networks, Astronomical Journal, 103: 308-331, 1992.

[100] U-M. O'Reilly and F. Oppacher, A comparative analysis of GP, In Peter J. Angeline and K. E. Kinnear, Jr., editors, Advances in Genetic Programming 2, ch. 2, 23-44. MIT Press, Cambridge, MA. 1996.

[101] B.H. Park and H. Kargupta, Distributed Data Mining, In The Handbook of Data Mining, Ed. Nong Ye, Lawrence Erlbaum Associates, 2003.

[102] Y-J. Park and M-S. Song, A genetic algorithm for clustering problems, In Proc. 3rd Annual Conf. on Genetic Programming, (1998) 568-575.

[103] C.H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, 1982

[104] R. Poli and J. Page, Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code GP and demes, Genetic Programming And Evolvable Machines, 1(1/2), 37-56, 2000.

[105] C. Procopiuc, M. Jones, P. Agarwal, and T. M. Murali, A Monte Carlo algorithm for fast projective clustering. In Proc. 2002 SIGMOD Conference, 418-427, 2002.

[106] P. Pudil, F. J. Ferri, J. Novovicova, and J. Kittler, Floating search methods for feature selection with nonmonotonic criterion functions. 12th International Conference on Pattern Recognition, 279-283, 1994.

[107] P. Pudil, J. Novovicová, Feature Subset Selection Using a Genetic Algorithm in Feature Extrac-tion, Construction and Selection: A Data Mining Perspective, Huan Liu, Hiroshi Motoda (Eds.), Kluwer, 1998.

[108] W.F. Punch, E.D. Goodman, M. Pei, L. Chia-Shun, P. Hovland and R. Enbody, Further Research on Feature Selection and Classification Using Genetic Algorithms, In Proc. 5th International Conference on Genetic Algorithms, Urbana-Champaign, 557-562, 1993.

[109] Y. Qian and C. Suen, Clustering Combination Method, International Conference on Pattern Recognition, ICPR'00, Volume 2, 2000.

[110] J. R. Quinlan, Bagging, boosting, and C4.5. In Proceedings of the 13th AAAI Conference on Artificial Intelligence, AAAI Press, Menlo Park, CA, 725-30, 1996.

[111] N.J. Radcliffe and P.D. Surry, Formal memetic algorithms, in ``Evolutionary Computing: AISB Workshop", Ed: T. Fogarty, Springer-Verlag, 1994.

[112] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn, and A. K. Jain, Dimensionality reduction using genetic algorithms, IEEE Transactions on Evolutionary Computation, 4(2): 164-171, 2000.

[113] M. Raymer, W. Punch, E. Goodman, P. Sanschagrin, and L. Kuhn, Simultaneous Feature Extrac-tion and Selection using a Masking Genetic Algorithm, In Proceedings of ICGA-97, pp. 561-567, San Francisco, CA, July, 1997.

[114] G.V. Reklaitis, A. Ravindran and K.M. Ragsdell, Engineering Optimization: Methods and Applications, Wiley, New York, 1983.

[115] K. Rodriguez-Vazquez, Identification of Non-Linear MIMO Systems Using Evolutionary Computation, Late Breaking Papers of Genetic and Evolutionary Computation Conf., 411-417, 2000.

[116] D.B. Rubin, Inference with Missing Data, Biometrika, 63: 581-592, 1976.

[117] J. C. Schlimmer, Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In Proceedings of the Tenth International Conference on Machine Learning, pages 284-290, 1993.

[118] W. Siedlecki and J. Sklansky, A note on genetic algorithms for large-scale feature selection, Pattern Recognition Letters, vol. 10, pp. 335-347, 1989.

[119] N. Slonim and N. Tishby, Agglomerative Information Bottleneck, In proceedings of NIPS-12, MIT Press, 617-623, 1999.

[120] N. Slonim, N. Friedman, and N. Tishby, Agglomerative Multivariate Information Bottleneck, In advances in Neural Information Processing Systems, NIPS-14, 2001.

[121] P. Somol, P. Pudil, F.J. Ferri and J. Kittler, Fast branch and bound algorithm in feature selection. World Multiconference on Systemics, Cybernetics and Informatics (SCI), Proceedings of SCI/ISAS 2000, Volume VII, (B. Sanchez, J.M. Pineda, J. Wolfmann, Z. Bellahsene, y F.J. Ferri, eds.), 646-651, 2000.

[122] K. Spark-Jones and D.M. Jackson, The use of automatically-obtained keyword classifications for information retrieval, Information Processing and Management 5, 175—201, 1970.

[123] S.D. Stearns, On Selecting Features for Pattern Classifiers. In Proc. 3d International Conference on Pattern Recognition, Coronado, CA, 678 - 686, 1978.

[124] A. Strehl and J. Ghosh, Cluster ensembles - a knowledge reuse framework for combining multiple partitions, Journal of Machine Learning Research, 3: 583-617, 2002.

[125] A. Strehl and J. Ghosh, Cluster ensembles - a knowledge reuse framework for combining partitionings. In Proc. Conference on Artificial Intelligence (AAAI 2002), Edmonton, 93-98. AAAI/MIT Press, 2002.

[126] N. Tishby, F.C. Pereira, and W. Bialek, The information bottleneck method, In Proc. of the 37-th Annual Allerton Conference on Communication, Control and Computing, 368-377, 1999.

[127] A. Thompson, Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution, Springer-Verlag: London, 1998.

[128] A. Topchy, B. Minaei, A.K. Jain, W. Punch, Adaptive Clustering Ensembles, In Proc. Intl. Conf on Pattern Recognition, ICPR 2004, Cambridge, UK, Aug 2004.

[129] A. Topchy, A.K. Jain and W. Punch, A Mixture Model for Clustering Ensembles, in Proc. SIAM Conf. on Data Mining, pp. 379-390, Orlando, 2004.

[130] A. Topchy, A.K. Jain and W. Punch, Combining Multiple Weak Clusterings, in Proceedings IEEE Intl. Conf. on Data Mining 2003, Melbourne, Fl, pp. 331-338, 2003.

[131] A. Topchy and W. Punch, Dimensionality Reduction via Genetic Value Clustering, Genetic Algorithms and Evolutionary Computation – GECCO 2003, Springer, LNCS 2724, 1431-1443, 2003

[132] A. Topchy and W. Punch, Faster Genetic Programming based on Local Gradient Search of Numeric Leaf Values, Genetic Algorithms and Evolutionary Computation - GECCO 2001, Morgan Kaufmann, pp. 155-162, 2001.

[133] G.V. Trunk, A problem of dimensionality: a simple example. IEEE Trans. Patt. Anal. Mach. Intell. 1, 306-307, 1979.

[134] P. Turney, How to shift bias: Lessons from the Baldwin effect, Evolutionary Computation, 4 (3), 271 - 295, 1996.

[135] H. Vafaie, K. DeJong, Feature Space Transformation Using Genetic Algorithms. IEEE Intelligent Systems 13 (2): 57 - 65, 1998.

[136] H. Vafaie, and K. DeJong, Robust feature selection algorithms, In Proc. of the 5th IEEE International Conference on Tools for Artificial Intelligence, Boston, MA, 356-363, 1993.

[137] J. M. Van Campenhout, On the peaking of the Hughes mean recognition accuracy: the resolution of an apparent paradox. IEEE Trans. on Systems, Man and Cybernetics, 8 (5): 390-395, 1978.

[138] J.A.M. Vermaseren, Tuning FORM with large calculations, Nucl.Phys.Proc.Suppl. 116, 343-347, 2003.

[139] P.M. Waser and C. Strobeck, Genetic signatures of interpopulation dispersal. Trends Ecol. Evol. 13, 43 - 44, 1998.

[140] A. Watson and I. Parmee, Systems Identification using Genetic Programming, In Proceedings of Int. Conf on Adaptive Computing in Engineering Design and Manufacture, 248 - 255, ACEDC'96, University of Plymouth, UK, 1996.

[141] D. Whitley, S. Gordon and K. Mathias, Lamarckian Evolution, The Baldwin Effect and Function Optimization. In Proceedings Parallel Problem Solving from Nature, PPSN III, 6-15. 1994.

[142] D. Whitley, R. Beveridge, C. Guerra, and C. Graves, Messy Genetic Algorithms for Subset Feature Selection. International Conference on Genetic Algorithms. T. Beck, ed. Morgan Kaufmann, 1997.

[143] D. H. Wolpert and W. G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, 67 - 82, 1997.

[144] S. Wright, The interpretation of population structure by F-statistics with special regards to system of mating. Evolution 19: 395 – 420, 1965.

[145] J. Yang, and V. Honavar, Feature Subset Selection Using a Genetic Algorithm, In: Feature Extraction, Construction, and Subset Selection: A Data Mining Perspective. Motoda, H. and Liu, H. (Eds.) New York, Kluwer, 1998.

[146] B. Yu and B. Yuan, A more efficient branch and bound algorithm for feature selection. Pattern Recognition, 26: 883 – 889, 1993.

[147] B. Zhang and H. Mühlenbein, Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor, Complex Systems, 7(3), 199 –220, 1993.

[148] A.Y. Zomaya, J.A. Andreson, D.B. Fogel, G.J. Milburn, and G. Rozenberg, Non-conventional Computing Paradigms in the New Millennium, IEEE/AIP Computing in Science and Engineering, November/December, Vol. 3, No. 6, 82 - 99, 2001.