2-2007

Clustering and combinatorial optimization in recursive supervised learning

Kiruthika RAMANATHAN
Singapore Management University, kiruthikar@smu.edu.sg

Sheng Uei GUAN

J Comb Optim (2007) 13:137–152
DOI 10.1007/s10878-006-9017-5

Clustering and combinatorial optimization in recursive supervised learning

Kiruthika Ramanathan · Sheng Uei Guan

Published online: 20 October 2006

Abstract The use of combinations of weak learners to learn a dataset has been shown to be better than the use of a single strong learner. In fact, the idea is so successful that boosting, an algorithm combining several weak learners for supervised learning, has been considered to be the best off the shelf classifier. However, some problems still exist, including determining the optimal number of weak learners and the over fitting of data. In an earlier work, we developed the RPHP algorithm which solves both these problems by using a combination of global search, weak learning and pattern distribution. In this chapter, we revise the global search component by replacing it with a cluster based combinatorial optimization. Patterns are clustered according to the output space of the problem, i.e., natural clusters are formed based on patterns belonging to each class. A combinatorial optimization problem is therefore created, which is solved using evolutionary algorithms. The evolutionary algorithms identify the "easy" and the "difficult" clusters in the system. The removal of the easy patterns then gives way to the focused learning of the more complicated patterns. The problem therefore becomes recursively simpler. Over fitting is overcome by using a set of validation patterns along with a pattern distributor. An algorithm is also proposed to use the pattern distributor to determine the optimal number of recursions and hence the optimal number of weak learners for the problem. Empirical studies show generally good performance when compared to other state of the art methods.

Keywords Topology based selection · Recursive learning · Task decomposition · Neural networks · Evolutionary algorithms

1 Introduction

Recursive supervised learners are a combination of *weak learners*, *data decomposition* and *integration*. Instead of learning the whole dataset, different *learners* (*neural networks*, for instance) are used to learn different subsets of the data, resulting in several *sub solutions*

K. Ramanathan (✉) · S. U. Guan
Department of Electrical and Computer Engineering, National University of Singapore,
10 Kent Ridge Crescent, Singapore 119260
e-mail: kiruthika_r@nus.edu.sg

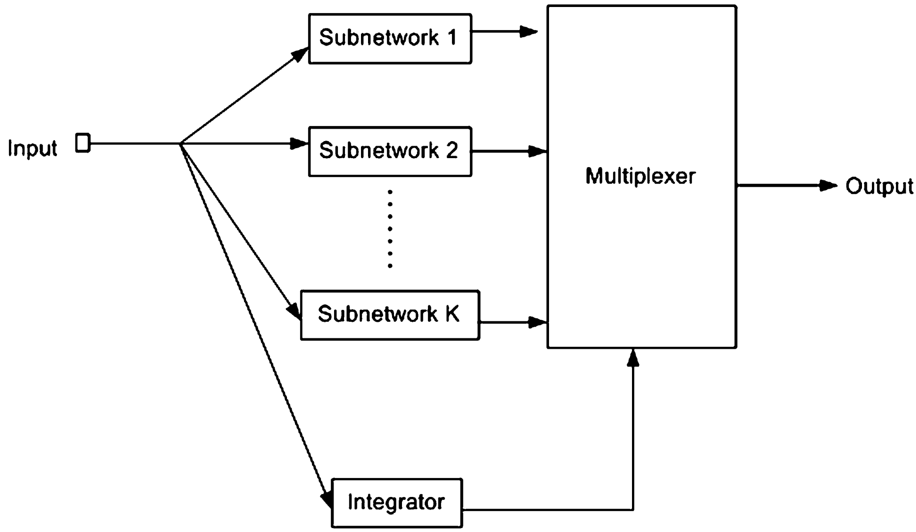


Fig. 1 Architecture of the final solution of recursive learners

(or *sub-networks*). These sub-networks are then integrated together to form the final solution to the system. Figure 1 shows the general architecture of such learners.

In the design of recursive learners, several factors come into play in determining the generalization accuracy of the system. Factors include

1. The accuracy of the sub networks
2. The accuracy of the integrator
3. The number of subnetworks

The choice of subsets for training each of the subnetworks plays an important part in determining the effect of all these factors. Various methods have been used for subset selection in literature. Several works, including topology based selection (Lasarzyck et al., 2004) and recursive pattern based training (Guan and Ramanathan, 2004), use evolutionary algorithms to choose subsets of data.

Algorithms such as active data selection (Zhang and Cho, 1998), boosting (Schapire, 1999) and multi-sieving (Lu et al., 1995) implement this subset choice using neural networks trained with various training methods. Multiple weak learners have also been used and the best weak learner given the responsibility of training and solving a subset (Guan et al., 2006). Other algorithms make use of more brute force methods such as the use of the Mahalanobhis distance (Fukunaga, 1990). Even other algorithms such as the output parallelism algorithm manually decompose their tasks (Guan and Li, 2002; Guan et al., 2004; Guan and Zhu, 2004). Clustering has also been used to decompose datasets in some cases (Engelbrechet, 2002).

The common method used in these algorithms (except the manual decomposition), is to allow a network to learn some patterns and declare these patterns as "*learnt*". The system then creates other networks to deal with the "*unlearnt*" patterns. The process is done recursively and with successively decreasing subset size, allowing the system to concentrate more and more on the "*difficult*" patterns.

While all these methods work relatively well, the hitch lies in the fact that, with the exception of manual partitioning, most of the techniques above use some kind of intelligent learner to split the data. While intelligent learners and algorithms such as neural networks (Haykins, 1999), genetic algorithms (Holland, 1973) etc. are effective algorithms, they are usually considered as black boxes (Dayhoff and Deleo, 2001), with little known about the structure of the underlying solution.

In this chapter, our aim is to reduce, by a certain degree, this black box nature of recursive data decomposition and training. Like in previous works, genetic algorithms are used to select subsets, however, unlike previous works; genetic algorithms are not used to select the patterns for a subset, but to select clusters of patterns for a subset. By using this approach, we hope to group patterns into subsets and derive a more optimal partitioning of data. We also aim to gain a better understanding of optimal data partitioning and the features of well partitioned data.

The system proposed consists of a *pre-trainer* and a *trainer*. The pre-trainer is made up of a *clusterer* and a *pattern distributor*. The clusterer splits the data set into clusters of patterns. The pattern distributor assigns validation patterns to each of these clusterers. The trainer now solves a combinatorial optimization problem, choosing the clusters that can be learnt with best training and validation accuracy. These clusters now form the "easy" patterns which re then learnt using a *gradient descent* algorithm to create the first subnetwork. The remaining clusters form the "difficult" patterns. The trainer now focuses attention on the difficult patterns, thereby recursively isolating and learning increasingly *"difficult"* patterns and creating several corresponding subnetworks.

The use of genetic algorithms in selecting clusters is expected to be more efficient than their use in the selection of patterns for two reasons

1. The number of combinations is now ${}^{n}C_k$ as opposed to ${}^{T}C_L$, where the number of available clusters n, is less than the number of training patterns T. Similarly, the number of clusters chosen k is is smaller than the number of training patterns chosen L. The solution space is now smaller, therefore increasing the probability of finding the better solutions.
2. The distribution of validation information is performed during pretraining, as opposed to during the training time. Validation pattern distribution is therefore a one-off process, thereby saving training time.

The rest of the paper is organized as follows. We begin with some preliminaries and related work in Section 2. In Section 3, we present a detailed description of the proposed *Recursive supervised learning with clustering and combinatorial optimization* (RSL-CC) algorithm. To increase the practical significance of the paper, we include pseudo code, remarks and parameter considerations where appropriate. More details and specifics of the algorithm are then presented in Section 4. In Section 5, we present some heuristics and practical guidelines for making the algorithm perform better. Section 6 presents the results of the RSL-CC algorithm on some benchmark pattern recognition problems, comparing them with other recursive hybrid and non-hybrid techniques. In Section 7, we complete the study of the RSL-CC algorithm. We summarize the important advantages and limitations of the algorithm and conclude with some general discussion and future work.

2 Some preliminaries

Notation

m: input dimension
n: output dimension

K: number of recursions
I: input
O: output
Tr: Training
Val: Validation
P: ensemble of subsets
S: Neural network solution
E: Error
T: number of training patterns
r: Recursion index
N$_{chrom}$: Number of chromosomes
N_c: Number of clusters

Problem formulation for recursive learning

Let $\mathbf{I_{tr}} = \{I_1, I_2, \ldots, I_T\}$ be a representation of T training inputs. I_j is defined, for any $j \in T$ over an m dimensional feature space, i.e, $I_j \in R^m$. Let $\mathbf{O_{tr}} = \{O_1, O_2, \ldots, O_T\}$ be a representation of the corresponding T training outputs. O_j is a binary string of length n. **Tr** is defined such that $\mathbf{Tr} = \{\mathbf{I_{tr}}, \mathbf{O_{tr}}\}$.

Similarly $\mathbf{I_v} = \{I_{v,1}, I_{v,2}, \ldots, I_{v,Tv}\}$ and $\mathbf{O_v} = \{O_{v,1}, O_{v,2}, \ldots, O_{v,Tv}\}$ represent the input and output patterns of a set of validation data, such that $\mathbf{Val} = \{\mathbf{I_v}, \mathbf{O_v}\}$. We wish to take Tr as training patterns to the system and Val as the validation data and come up with an ensemble of K subsets. Let **P** represent this ensemble of K subsets

$$\mathbf{P} = \{\mathbf{P^1}, \mathbf{P^2}, \ldots \mathbf{P^K}\}, \quad \text{where, for } i \in K$$
$$\mathbf{P^i} = \{\mathbf{Tr^i}, \mathbf{Val^i}\}, \quad \mathbf{Tr^i} = \left\{\mathbf{I^i_{tr}}, \mathbf{O^i_{tr}}\right\}, \quad \mathbf{Val^i} = \left\{\mathbf{I^i_v}, \mathbf{O^i_v}\right\}$$

Here, I^i_{tr} and O^i_{tr} are $m \times T^i$ and $n \times T^i$ matrices respectively and $\mathbf{I^i_v}$ and $\mathbf{O^i_v}$ are $m \times Tv^i$ and $n \times Tv^i$ matrices respectively, such that $\sum_{i=1}^K T^i = T$ and $\sum_{i=1}^K Tv^i = Tv$.

We need to find a set of neural networks $S = \{S^1, S^2, \ldots S^K\}$, where S^1 solves $\mathbf{P^1}$, S^2 solves $\mathbf{P^2}$ and so on.
P should fulfill two conditions:

Conditional set 1

1. The individual subsets can be trained with a small mean square training error, i.e,

$$E^i_{\mathrm{tr}} = \left\|\mathbf{O^i_{tr}} - S^i\left(\mathbf{I^i_{tr}}\right)\right\| \longrightarrow 0,$$

2. None of the subsets $\mathbf{Tr^1}$ to $\mathbf{Tr^K}$ are overtrained, i.e, $\sum_i^j E^i_{\mathrm{val}} < \sum_{i=1}^{j+1} E^i_{\mathrm{val}}$; $j, j+1 \in K$

Related work

As mentioned in the introduction, several methods are used to select a suitable partition $\mathbf{P} = \left\{\mathbf{P^1}, \mathbf{P^2}, \ldots \mathbf{P^K}\right\}$ that fulfils the conditions 1 and 2 above. In this section we shall discuss some of them in greater detail and discuss their pros and cons

Manual decomposition

Output parallelism was developed by Guan and Li (2002). The idea involves splitting a n-class problem into n two-class problems. The idea behind the decomposition is that a two class problem is easier to solve than an n-class problem and is hence a more efficient algorithm. Guan et al. (2004) later added an integrator in the form of a pattern distributor to the system. The Output parallelism algorithm essentially develops a set of n sub-networks, each catering to a 2-class problem and integrates them using a pattern distributor.

While the algorithm is shown to be effective in terms of both training time and generalization accuracy, a major drawback of the algorithm is its class based manual decomposition. In fact, research has been carried out (Guan et al., 2004) shows empirically that the 2-class decomposition is not necessarily the optimum decomposition for a problem. This optimum decomposition is a problem dependent value. While automatic algorithms have been developed to overcome this problem of manual decomposition (Guan et al., 2005), the net result is an algorithm which is computationally expensive. The other concern associated with the output parallelism is that it can only be applied to classification problems.

Genetic algorithm based partitioning algorithms

GA based algorithms are shown to be effective in partitioning the datasets into simpler subsets. One interesting observation was that using GA for partitioning leads to simpler and easily separable subsets (Guan and Ramanathan, 2004).

However, the problem with these approaches is that the use of GA is computationally costly. Also, a criterion has to be established to separate the difficult patterns from the first, and in the case of various algorithms, criterions used include the history of difficulty, the degree of learning (Lasarzyck et al., 2004) etc.

In the Recursive Pattern based hybrid supervised learning (RPHS) algorithm (Guan and Ramanathan, 2004), the problem of the degree of learning is overcome by hybridizing the algorithm and adapting the solution to the problem topology by using a neural network. In this algorithm, GA is only a pattern selection tool. However, the problem of computational cost still remains.

Brute force methods

Brute force mechanisms include the use of a distance or similarity measure such as the Mahalanobhis distance (Fukunaga, 1990). Subsets are selected to ensure good separation of patterns in one output class from another, resulting in good separation between classes in each subset. A similar method, bounding boxes, has been developed (Guan et al., 2006) which clusters patterns based on their Euclidean distances and overlap from patterns of other classes.

While the objective of the approach is direct, much effort in involved in pretraining as it involves brute force distance computation. In Guan et al. (2006), the authors have shown that the complexity of the brute force distance measure increases almost exponentially with the problem dimensionality. Also, a single distance based criterion may not be the most suitable for different problems, or even different classes.

Neural network approaches

Neural network approaches to decomposition are similar to genetic algorithms based approaches. Multisieving (Lu et al., 1995), for instance, allows a neural network to learn patterns, after which it isolates the network and allows another network to learn the "unlearnt" patterns. The process continues until all the patterns are learnt.

Boosting (Schapire, 1999) is similar in nature, except that instead of isolating the "learnt" patterns, a weight is assigned to them. An unlearnt pattern has more weight and is thus concentrated on by the new network. Boosting is a successful method and has been regarded as one of the best off the shelf classifiers" in the world (Hastie et al., 2001).

However, the system uses neural networks to select patterns, which solves the problem, but results in the black box. The reasons why some patterns are learnt better is unknown, and while one can solve the dataset with these approaches, not much information is gained about the data in the solving process. Also, the multisieving algorithm, for instance, does not talk about generalization, which is often an issue in task decomposition approaches. Subsets which are too small can result in the overtraining of the corresponding subnetwork.

Clustering based approaches

Clustering based approaches to task decomposition for supervised learning are many fold (Engelbrechet and Brits, 2002). Most of them divide the dataset into clusters and solve individual clusters separately. However, we feel that this particular approach is not very good as it creates a bias. Many subsets have several patterns in one class and few patterns in other classes. The networks created are therefore not sufficiently robust.

3 The RSL-CC algorithm

The RSL-CC algorithm can be described in two parts, *pre-training* and *training*. In this section, we explain these two aspects of training in detail

Pretraining

1. We express $\mathbf{I_{tr}}$, $\mathbf{O_{tr}}$, $\mathbf{I_v}$ and $\mathbf{O_v}$ as a combination of m classes of patterns, i.e,

$$\mathbf{I_{tr}} = \{\mathbf{I^{C1}}, \mathbf{I^{C2}}, \ldots \mathbf{I}^{\mathbf{C}n}\}$$
$$\mathbf{O_{tr}} = \{\mathbf{O^{C1}}, \mathbf{O^{C2}}, \ldots \mathrm{O}^{\mathbf{C}n}\}$$
$$\mathbf{I_v} = \left\{\mathbf{I_v^{C1}}, \mathbf{I_v^{C2}}, \ldots, \mathbf{I}_\mathbf{v}^{\mathbf{C}n}\right\}$$
$$\mathbf{O_v} = \left\{\mathbf{O_v^{C1}}, \mathbf{O_v^{C2}}, \ldots, \mathbf{O}_\mathbf{v}^{\mathbf{C}n}\right\}$$

2. The datasets $\mathbf{I_{tr}}$, $\mathbf{O_{tr}}$, $\mathbf{I_v}$ and $\mathbf{O_v}$ are split into n subsets,

$$\left\{\mathrm{I^{C1}}, \mathbf{O^{C1}}, \mathbf{I_v^{C1}}, \mathbf{O_v^{C1}}\right\}, \left\{\mathrm{I^{C2}}, \mathbf{O^{C2}}, \mathbf{I_v^{C2}}, \mathbf{O_v^{C2}}\right\}, \ldots \left\{\mathrm{I}^{\mathbf{C}n}, \mathbf{O}^{\mathbf{C}n}, \mathbf{I}_\mathbf{v}^{\mathbf{C}n}, \mathbf{O}_\mathbf{v}^{\mathbf{C}n}\right\} \tag{1}$$

   where each subset in expression (1) consists of only patterns from one class.
3. Each subset, $\{\mathbf{I}^{\mathbf{C}i}, \mathbf{O}^{\mathbf{C}i}, \mathbf{I}_\mathbf{v}^{\mathbf{C}i}, \mathbf{O}_\mathbf{v}^{\mathbf{C}i}\}$, $i \in n$, now undergoes a clustering treatment as below:

   - Cluster $\mathbf{I}^{\mathbf{C}i}$ into k^{Ci} partitions or *natural clusters*. In this paper, we obtain these natural clusters by using Agglomerative Hierarchical Clustering (Jain et al., 1999).

- Using a pattern distributor, patterns in $\mathbf{I}_{\mathbf{v}}^{\mathbf{C}i}$ are assigned to one of the k^{Ci} patterns. In this paper, we implement the pattern distributor using the nearest neighbor algorithm (Wong and Lane, 1983).
- Each pattern, validation or training, in a given cluster j^{Ci}, $j \in k$, has the same output pattern.

4. The total number of clusters is now the sum of the natural clusters formed in each class.

$$N_c = \sum_{i=1}^{n} k^{ci} \tag{2}$$

Training

1. Number of recursions $r = 1$
2. A set of binary chromosomes are created, each chromosome having N_c elements, where N_c is defined as in (2).
   An element in a chromosome is set at 0 or 1, 1 indicating that the corresponding cluster will be selected for solving using recursion r.
3. A genetic algorithm is executed to minimize E_{tot}, the average of the training and validation errors E_{Tr} and E_{Val}

$$E_{\text{tot}} = \frac{1}{2}(E_{\text{Tr}} + E_{\text{Val}}) \tag{3}$$

4. The best chromosome $Chrom_{best}$ is a binary string with a combination of 0s and 1s, with the size N_c. The following steps are executed

   a. $N_c^r = 0$, $\mathbf{Tr^r} = []$, $\mathbf{Val^r} = []$
   b. For $e = 1$ to N_c
      i. if $Chrom_{best}(e) = 1$

$$N_c^r ++$$

$$\mathbf{Tr^r} = \mathbf{Tr^r} + \mathbf{Tr_{chrom_{best}(e)}}$$

$$\mathbf{Val^r} = \mathbf{Val^r} + \mathbf{Val_{chrom_{best}(e)}}$$

   b. The data is updated as follows:

$$\mathbf{Tr} = \mathbf{Tr} - \mathbf{Tr^r}$$

$$\mathbf{Val} = \mathbf{Val} - \mathbf{Val^r}$$

$$N_c = N_c - N_c^r$$

$$r ++$$

   c. $\mathbf{Tr^r}$ and $\mathbf{Val^r}$ are used to find S^r, the solution network corresponding to the subset of data in recursion r.

5. Steps 2 to 4 are repeated with the new values of **Tr**, **Val**, N_c and r.

Simulation

Simulating and testing the RSL-CC algorithm is implemented using a Kth nearest neighbor (KNN) (Wong and Lane, 1989) based pattern distributor. KNN was used to implement the pattern distributor due to the ease of its implementation. At the end training, we have K subsets of data. A given test pattern is matched with its nearest neighbor. If the neighbor belongs to subset i, the pattern is also deemed as belonging to subset i. The solution for subset i is then used to find the output of the pattern. A multiplexer is used for this function. The KNN distributor provides the select input for the multiplexer, while the outputs of subnetworks 1 to K are the data inputs. This process is illustrated by Fig. 4.

4 Details of the RSL-CC algorithm

Figure 3 summarizes the data decomposition of the RSL-CC algorithm. Hypothetically, we can observe the algorithm as finding the simpler subset of data and developing a subnetwork to solve the subset. The size of the "complicated" subset becomes smaller as training proceeds, thereby allowing the system to focus more on the "complicated" data. When the size of the remaining dataset becomes too small, we find that there is no motivation for further decomposition and the remaining data is trained in the best possible way. Later in this paper, we observe how the use of GA's combinatorial optimization automatically takes care of when to stop recursions. The use of GAs to select patterns (Lasarzyck et al., 2004; Guan and Ramanathan, 2004) requires extensive tests for detrimental decomposition and overtraining. The proposed RSL-CC algorithm eliminates the need for such tests. As a result, the resulting algorithm is self sufficient and very simple, with minimal adaptations.

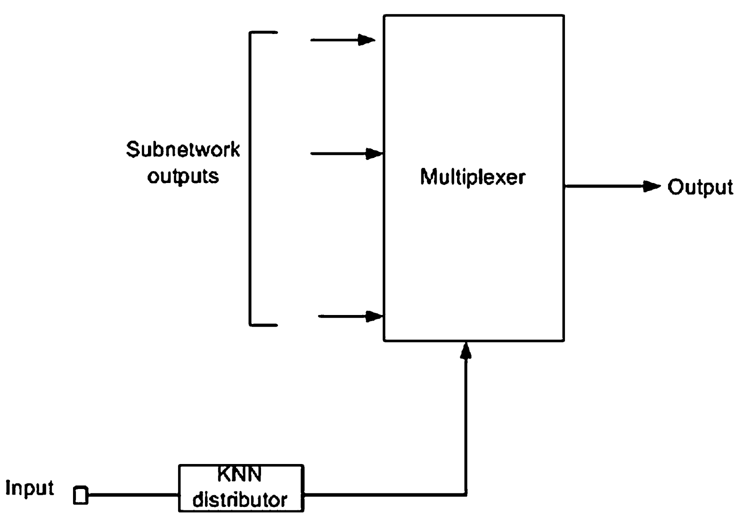


Fig. 2 Using a KNN distributor to test the RSL system

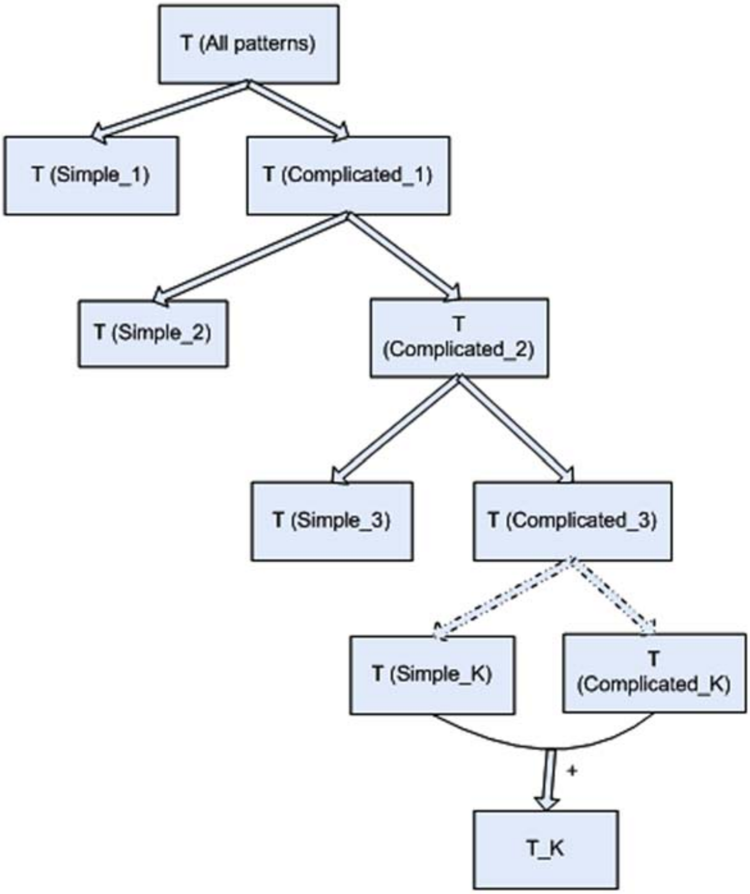


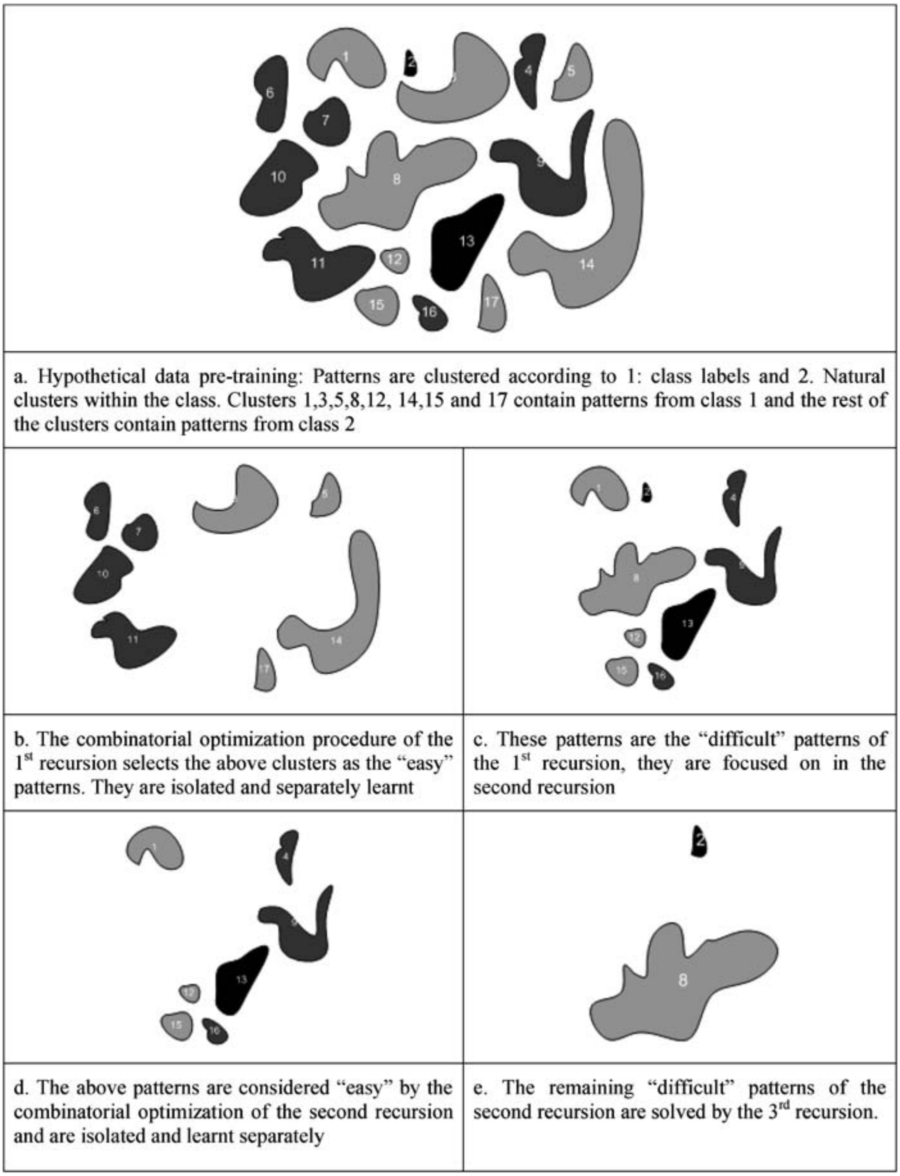
Fig. 3 Illustration of the data decomposition of the RSL-CC algorithm

Illustration

Figure 4 shows a hypothetical condition where the RSL-CC algorithm is applied to create a system to learn the dataset shown. The steps performed on the dataset are traced below. With the data in Fig. 4, the best chromosome selected at the end of the first recursion has the configuration: **"0 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1"**, the chromosome selected at the end of the second recursion has a configuration: **"1 0 1 0 1 1 1 1 1"**. And at the end of the third recursion, the chromosome has the configuration **"1 1"**. All the remaining data is selected and the training is considered complete.

Termination criteria

The grouping of patterns means that clusters of patterns are selected for each subset. Further, in contrast with any other method, the GA based recursive subset selection proposed selects the optimal subset combination. Therefore, we can assume that the decomposition performed is optimal. We prove this by using apogage



a. Hypothetical data pre-training: Patterns are clustered according to 1: class labels and 2. Natural clusters within the class. Clusters 1,3,5,8,12, 14,15 and 17 contain patterns from class 1 and the rest of the clusters contain patterns from class 2

b. The combinatorial optimization procedure of the 1st recursion selects the above clusters as the "easy" patterns. They are isolated and separately learnt

c. These patterns are the "difficult" patterns of the 1st recursion, they are focused on in the second recursion

d. The above patterns are considered "easy" by the combinatorial optimization of the second recursion and are isolated and learnt separately

e. The remaining "difficult" patterns of the second recursion are solved by the 3rd recursion.

Fig. 4 Hypothetical application of RSL-CC, with steps traced

Proof: Apogage is used.

If the subset chosen at recursion i is not optimal, an alternative subset will be chosen. The largest possible alternative subset is the training set for the recursion, $\mathbf{Tr^r} = \mathbf{Tr} - \sum_{i=1}^{r-1} \mathbf{Tr^i}$. Therefore is decomposition at a particular recursion is suboptimal, no decomposition will be performed and the training will complete.

We therefore have the following termination conditions

Condition set 2: Termination conditions
Condition 1: No clusters of patterns are left in the system.
Condition 2: Only one cluster is left in the remaining data.
Condition 3: More than one cluster is present in the remaining data, but all the clusters belong to the same class.

Condition 1 occurs when the optimal choice in a system is to choose all the clusters as decomposition is not favorable. Conditions 2 and 3 describe dealing with cases when it is not necessary to create a classifier due to the homogeneity of output classes.

Fitness function for combinatorial optimization

In Eq. (3), we defined the fitness function as an average of the training and validation errors obtained when training the subset selected by the chromosome, $\frac{1}{2}(E_{\mathrm{Tr}} + E_{\mathrm{Val}})$. The values of E_{Tr} and E_{Val} are calculated as follows

1. Design a 3 layered neural network with an arbitrary number of hidden nodes (we use 10 nodes, for the purpose of this paper).
2. Use the Training and Validation subsets selected by the corresponding chromosome to train the network.

The best performing network is chosen as $Chrom_{best}$

5 Heuristics for improving the performance of the RSL-CC algorithm

We propose here several methods which will improve and make the algorithm "to the point" as far as implementation is concerned.

Population size

The number of elements in each chromosome depends on the total number of cluster formed. However, the number of chromosomes in the population, in this paper, is evaluated as follows

$$N_{chorm} = \min(2^{N_c}, POP\_SIZE) \tag{4}$$

This means that the population size is either POP_SIZE, a constant for the maximal population size, or if N_c is small, 2^{N_c}.

The argument behind the use of a smaller population size is so that when there are 4 clusters, for example, it does not make much efficiency to evaluate a large number of chromosomes. So only 16 chromosomes are created and evaluated.

Number of generations

In the case where the number of chromosomes is 2^{N_c}, only one generation is executed. This step is again to ensure efficiency of the algorithm.

Duplication of chromosomes: Again, with efficiency in mind, we ensure that in the case where the population size is 2^{N_c}, we ensure that all the chromosomes are unique. Therefore, when the number of clusters is small, the algorithm is a brute force technique.

6 Experimental results

Problems considered

The table below summarizes the three classification problems considered in this paper. The problems were chosen such that they varied in terms of input and output dimensions as well as in the number of training, testing and validation patterns made available. All the datasets, other than the two-spiral dataset, were obtained from the UCI repository.

The results of the two-spiral dataset were compared with constructive backpropagation, multisieving and the topology based subset selection algorithms only. This was because the SPAM and two spiral problems were two-class problems. Therefore implementing the output parallelism will not make a difference to the results obtained by CBP.

The two spiral dataset consists of 194 patterns. To ensure a fair comparison to the Dynamic subset selection algorithm (Lasarzyck et al., 2004), test and validation datasets of 192 patterns were constructed by choosing points next to the original points in the dataset as mentioned in the paper.

Experimental parameters and control algorithms implemented

Table 2 summarizes the parameters used in the experiments. As we wish for the RSL-CC technique to be as problem independent as possible, we make all the experimental parameters

Table 1 Summary of the problems considered

Problem Name	VOWEL	LETTER RECOGNITION	TWO SPIRAL
Training set size	495	10000	194
Test set size	248	5000	192
Validation set size	247	5000	192
Number of Inputs	10	16	2
Number of Outputs	11	26	2

Table 2 Summary of parameters used

Parameter		Value
Agglomerative clustering parameters	Distance metric	Cityblock
	Linkage measure	Furthest distance
	Cutoff depth	2 (MATLAB default value)
Evolutionary search parameters	Population Size	50
	Crossover probability	0.9
	Small change mutation probability	0.1
Neural network parameters	Backpropagation learning rate	10^{-2}
	Number of neighbors in the KNN pattern distributor	1

Table 3 Summary of the results obtained from the VOWEL problem (38 clusters, 12 recursions)

Algorithm used	Training time (s)	C. error (%)
Constructive backpropagation	237.9	37.16
Multisieving with KNN pattern distributor	318.23	39.43
Output parallelism	418.9	25.54
Output parallelism with pattern distributor	534.3	24.89
RPHS	*473.88*	*17.733*
Single clustering	458.43	25.24
RSL-CC	547.37	9.84

constant for all problems and as given below. Each experiment was run 40 times, with 4-fold cross validation.

For comparison purposes, the following algorithms were designed and implemented. The constructive backpropagation algorithm (Lehtokangas, 1999) was implemented as a single staged (non hybrid) algorithm which conducts gradient descent based search with the possibility of evolutionary training by the addition of a hidden node. The Multisieving algorithm (Lu et al., 1995) (recursive non hybrid pattern based selection) was implemented to show the necessity to find the correct pseudo global optimal solution.

The following control experiments were carried out based on the multisieving algorithm and the dynamic topology based subset finding algorithm. Both the versions of output parallelism implemented also show the effect of hybrid selection.

In order to illustrate the effect of the GA based combinatorial optimization, we also implement the single cluster algorithm explained in Section 2 (Engelbrechhet and Brits, 2002). The algorithm, in contrast to the RSL-CC algorithm, simply divides the data into clusters and develops a network to solve each cluster separately.

The RSL-CC algorithm was also compared to our earlier work on RPHS (Guan and Ramanathan, 2004), which uses a hybrid algorithm to recursively select patterns, as opposed to clusters.

In a nutshell, the following algorithms were implemented to compare with the various properties of the RSL-CC algorithm

1. Constructive backpropagation
2. Multisieving[1]
3. Dynamic topology based subset finding
4. Output parallelism without pattern distributor (Guan and Li, 2002)
5. Output parallelism with pattern distributor (Guan et al., 2004)
6. Single clustering for supervised learning
7. Recursive pattern based hybrid supervised learning

Table 2 summarizes the parameters used. Using thresholding, the natural clusters of patterns in each class were obtained. AHC was preferred to other clustering methods such as K-means or SOMs due to its parametric nature and since the number of target clusters is not required beforehand. The simulations were conducted with MATLAB 7.0.

[1] The multisieving algorithm (Lu et al., 1995) did not propose a testing system. We are testing the generalization accuracy of the system using the KNN pattern distributor, similar to the RSL-CC pattern distributor.

Results

We divide this section into two parts. In the first part, we compare the mean generalization accuracies of the various recent algorithms described in this paper with the generalization accuracy of the RSL-CC algorithm. In the second part, we present the clusters and data decomposition employed by RSL-CC, illustrating the finding of simple subsets.

Comparison of generalization accuracies

From the tables above, we can observe that the generalization error of the RSL-CC is comparable to the generalization error of RPHS algorithm and is a general improvement over other recent algorithms. A particularly significant improvement can be observed in the vowel dataset.

There is some tradeoff observed in terms of training time. The training time for the RSL-CC algorithm for the vowel and two-spiral problems are higher than other methods. However, it is interesting to note that the training time for the Letter Recognition problem is more than 50% less than any of the recent algorithms. It is felt that this reduction is training time comes from the reduction of the problem space from the selection of patterns to the selection of clusters, where clusters are selected from 100 possible clusters while RPHS has to select patterns out of 10,000, thereby reducing the solution space by 100 fold.

On the other hand, for the vowel problem, the problem space is reduced by only about 13 fold. The performance of RSL-CC is more efficient when the reduction of the problem space is more significant than the GA-based combinatorial optimization.

Table 4 Summary of the results obtained from the LETTER RECOGNITION problem (100 clusters, 16 recursions)

Algorithm used	Training time (s)	C. error (%)
Constructive backpropagation	20845.05	21.672
Multisieving with KNN pattern distributor	55349	65.04
Output parallelism	42785.4	20.06
Output parallelism with pattern distributor	45625.4	18.636
RPHS	***29701***	***12.42***
Single clustering	NA	***NA***
RSL-CC	12682	***13.04***

Table 5 Summary of the results obtained from the TWO-spiral problem (4 clusters, 2 recursions)

Algorithm used	Training time (s)	C. error (%)
Constructive backpropagation	15.58	49.38
Multisieving with KNN pattern distributor	**35.89**	23.61
Dynamic Topology Based subset selection (TSS)	–	***28.0***
RPHS	***59.97***	***11.08***
Single clustering	14.35	**10.82**
RSL-CC	30.61	**10.82**

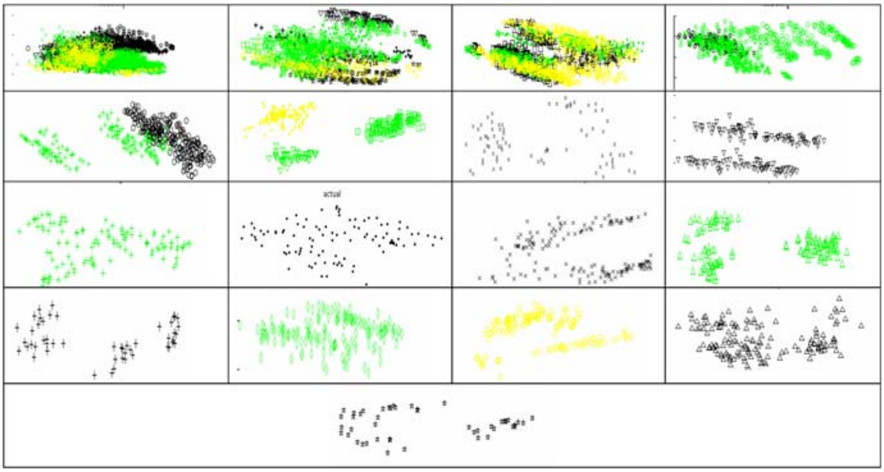


Fig. 5 Decomposition of data for the letter recognition problem

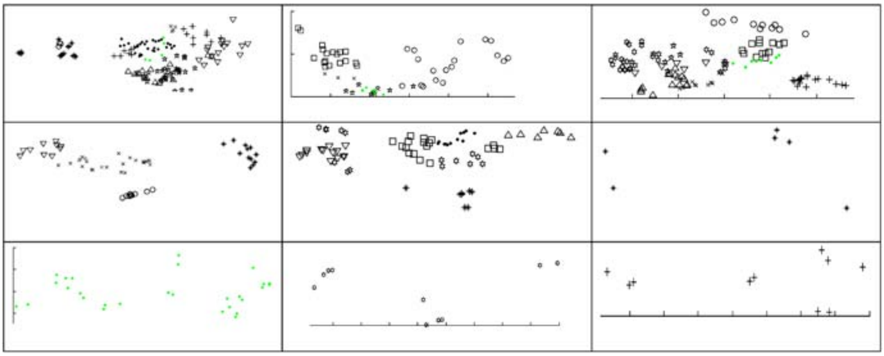


Fig. 6 Decomposition of data for the vowel problem

The RSL-CC decomposition figures

Figures 5 and 6 illustrate the data decomposition for the letter decomposition for the letter recognition and the vowel problems. Only one instance of decomposition is presented in the figures. From the figures, we can observe the data being split into increasingly smaller subsets, thereby increasing focus on the difficult patterns. The decomposition presented is the 2 dimensional projections on the principal component axis (PCA) (Fukunaga, 1991) of the input space.

7 Conclusions and future directions

In this chapter, we present the RSL-CC algorithm which divides the problem space into class based clusters, where a combination of clusters will form a subset. The problem therefore becomes a combinatorial optimization problem, where the clusters chosen for the subset becomes the parameter to be optimized. Genetic algorithms are used to solve this problem to select a good subset.

The subset chosen is then trained separately, and the combinatorial optimization problem is repeated with the remaining clusters. The situation progresses recursively until all the patterns are learnt. The sub networks are then integrated using a KNN based pattern distributor and a multiplexer.

Results show that reducing the problem space into clusters simplifies the problem space and produces generalization accuracies which are either comparable to or better than other recent algorithms in the same domain.

Future directions would include parallelizing the RSL-CC algorithm and exploring the use of other clustering methods such as K-means or SOMs on the algorithm. The study of the effect of various clustering algorithms will help us determine better the algorithm simplicity and the robustness. Also to be studied and determined are methods to further reduce the training time of combinatorial optimization, alternative fitness functions and ways to determine the robustness of class based clustering.

References

Dayhoff, DeLeo (2001) Artificial neural networks: Opening the black box. Cancer 91(8):1615–1635

Engelbrechet AP, Brits R (2002) Supervised training using an unsupervised approach to active learning. Neur Proc Lett 15(3):247–260

Fukunaga K (1990) Introduction to statistical pattern recognition. Academic Press, Boston

Guan SU, Li S (2002) Parallel growing and training of neural networks using output parallelism. IEEE Trans Neur Netw 13(3):542–550

Guan SU, Ramanathan K (2004) Recursive percentage based hybrid pattern training. In: Proc of IEEE conf on cybernetics and intelligent systems, pp. 455–460

Guan SU, Neo TN, Bao C (2004) Task decomposition using pattern distributor. J Intell Syst 13(2):123–150

Guan SU, Qi Y, Tan SK, Li S (2005) Output partitioning of neural networks. Neurocomputing 68:38–53

Guan SU, Ramanathan K, Iyer LR (2006) Multi learner based recursive training. In: Proc of IEEE conf on cybernetics and intelligent systems (Under review)

Guan SU, Zhu F (2004) Class decomposition for GA-based classifier agents—a pitt approach. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 34(1):381–392

Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning: data mining, inference, and prediction, Springer Series in Statistics Springer-Verlag

Haykins S (1999) Neural networks, a comprehensive foundation. Prentice Hall

Holland JH (1973) Genetic algorithms and the optimal allocation of trials. SIAM Journal on Computing 2(2): 88–105

MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of 5-th Berkeley symposium on mathematical statistics and probability. Berkeley, University of California, vol. 1, pp. 281–297

Jain AK, Murty MN, Flynn PJ (1999), Data clustering: a review. ACM Comput Surv 31(3):264–323

Kohonen T (1997) Self organizing maps. Springer-Verlag, Berlin

Lasarzyck CWG Dittrich P, Banzhaf W (2004) Dynamic subset selection based on a fitness case topology. Evol Comput 12(4):223–242

Lehtokangas (1999) Modeling with constructive Backpropagation. Neur Netw 12:707

Lu BL, Ito K, Kita H, Nishikawa Y(1995) Parallel and modular multi-sieving neural network architecture for constructive learning. In: Proceedings of the 4th international conference on artificial neural networks, vol. 409, pp. 92–97.

Schapire RE (1999) A brief introduction to boosting. In: Proceedings of the sixteenth international joint conference on artificial intelligence

Wong MA, Lane T (1983) A kth nearest neighbor clustering procedure. J Royal Statist Soc (B), 45(3):362–368

Zhang BT, Cho DY (1998) Genetic programming with active data selection. Lecture notes in computer science 1585:146–153