"COMPO"

CONCEPTUAL CLUSTERING
with
COMPETITIVE LEARNING and the GENETIC ALGORITHM

Hugo de Garis
CADEPS (CAIRU), CADEPS Artificial Intelligence Research Unit,
Université Libre de Bruxelles (U.L.B.), Ave. F.D. Roosevelt 50,
C.P.194/7, B-1050, Brussels, Belgium, Europe.
tel. +32 2 642 2783, email CADEPS@BRRNF11(.BITNET)

and

Machine Learning and Inference Laboratory,
Artificial Intelligence Center, George Mason University,
4400 University Drive, Fairfax, Virginia 22030, U.S.A.
tel. +1 703 764 6057, email HUGODEG@GMUVAX2.GMU.EDU

Keywords :
Conceptual Clustering, Descriptive Clustering, Connectionism, Competitive
Learning, Machine Learning, Data Analysis, Unsupervised Learning, Genetic Algorithm,
Partition Quality Measure, Classification Tree.

Abstract :

This paper introduces the idea that conceptual clustering can be performed using
connectionist competitive learning. Competitive learning is used to detect clusters of
objects and their corresponding (qualitative) descriptions. A genetic algorithm is employed
to choose a subset of these descriptions such that the objects matching them form partitions
over the population of objects concerned. Hierarchical classification trees are built by
recursing the above two steps (competitive learning "clustering" and genetic algorithm
"partitioning") over the objects matching the descriptions at each node.

Introduction :

This paper introduces the idea that conceptual clustering can be performed using
connectionist competitive learning [RUMELHART et al 1985]. Conceptual clustering
[MICHALSKI et al 1984] uses concept descriptions to perform the clustering of objects
into groups with high intracluster and low intercluster similarities, as distinct from the usual
numerical similarity measures between object descriptions in traditional cluster analysis
[EVERITT 1980]. In conceptual clustering it is possible that two objects may be placed
into two different clusters because they "match" two different concept descriptions, even
though they may be very close in numerical "distance" similarity terms.

However the concept of conceptual clustering used in this paper differs somewhat
from that of Michalski's. Here the term is used in a looser sense, namely that the clusters
detected are provided with a concept description (i.e. a description which the elements of
the cluster all satisfy). Michalski's definition is more specific. He uses concept descriptions
as a means to form the clusters in the first place. Still, the looser definition used here differs
greatly from the usual cluster analysis approach, where no cluster descriptions are given.
Perhaps "descriptive clustering" would be a better label for this looser sense.
The program described in this paper, called "COMPO", uses a competitive learning technique to detect clusters of objects which have common values of attributes. Hidden amongst these clusters are subsets which form partitions over the objects. A genetic algorithm [HOLLAND 1986, GOLDBERG 1989] is used to find these subsets. This two-step process (detection of clusters with competitive learning, followed by a selection of subsets of these clusters to form partitions over the objects concerned) is applied recursively to each cluster (node) to form a hierarchical classification tree.

Section 1 of this paper gives a brief description of the principles of connectionist competitive learning. Section 2 shows how COMPO uses competitive learning to detect intrinsic clusters of objects which have qualitative data descriptions. Section 3 gives a brief description of the principles of the genetic algorithm. Section 4 shows how COMPO uses the genetic algorithm to select subsets of the clusters such that the subsets form partitions over the objects concerned. Section 5 discusses the application of the above ideas to a simple data base. Section 6 shows how COMPO forms hierarchical classification trees. Section 7 presents some results taken from a larger data base. Section 8 contains the conclusion and suggestions for future research.

1. Connectionist Competitive Learning :

   Competitive Learning [VON DER MALSBURG 1973, RUMELHART et al 1985, GROSSBERG 1976, KOHONEN 1982], is a branch of Connectionism (the computational model which uses a large number of simple, highly connected, processing units (artificial neurons), [RUMELHART et al 1986]). Competitive learning is unsupervised. There is no human expert to classify a given input signal or vector. The weights (reflecting the strength of connections between neurons) are modified depending on the signals they receive from their input connections. There are several forms of competitive learning, but they all use the principle that the neuron which has maximum output for a given input signal has its weights modified. Neurons thus compete with each other in order to have a maximum output. Some forms of competitive learning also modify neighbouring neurons as well.

   If both the input signals and the weight vectors are normalised, i.e. the sum of the squares of the numerical components is made equal to 1, then competitive learning shifts the weight vector of the winning neuron towards the "center of gravity" of a cluster of similar input vectors. Another way of looking at this idea is that the weight vector becomes more similar to the input vector which causes the neuron to win. Thus competitive learning is a way of discovering features of object descriptions which can be used to classify a set of objects.

   Before the winning neuron can be found, the outputs of all the neurons need to be calculated. The output of a neuron (in this paper) is the simple numeric "dot product" of the input vector and the weight vector (see FIG. 1). In COMPO, there are only two layers of neurons, the input and output layers, and all input neurons connect to all the output neurons. The network is thus fully connected. The learning of the neural network occurs only on the output neurons. Once the output neuron with the highest output has been found, its weights are updated in the following way. A small fraction (say 5%) of the value of each weight is subtracted and redistributed evenly amongst the active connections (that is, those connections between the input neurons and the winning output neuron which have a non zero signal, as shown in FIG. 2). This fraction is called the "learning constant".
In this way the active connections are strengthened and the non active connections are weakened, so that the weight values approach those of the input vectors. For example, if a certain input neuron is active for a given subset of (similar) objects, then the weight of the connection between that input neuron and the output neuron which regularly wins for that subset of input vectors, will be strong. The weights on the output neurons can thus be used to detect and define clusters of object descriptions.

Out1 wins

Out1's weight vector changes from (0.2, 0.3, 0.5) to

\[(0.2 - 0.05*0.2 + 0.05/2, \ 0.3 - 0.05*0.3, \ 0.5 - 0.05*0.5 + 0.05/2)\]
\[= (0.215, \ 0.285, \ 0.5)\]
2. Detecting Intrinsic Clusters:

This section describes how competitive learning is used to discover intrinsic clusters of object descriptions. Before COMPO can be used, the input data has to be specially formatted. COMPO assumes that object descriptions are vectors of qualitative attribute-value pairs, e.g., ((age=young)(height=tall)(hair=brown)). For each different qualitative value of an attribute, there is a corresponding input neuron. Hence the number of neurons in the input layer is equal to the total number of qualitative values for the objects.

The set of different qualitative values for a given attribute are coded numerically from 1 to N (where N will vary from attribute to attribute), hence N of the input neurons are used to represent the values of the given attribute. If, for example, for a given input vector, a given attribute takes its third value, then the third of the N input neurons corresponding to that attribute will have an output of 1. The other N-1 input neurons will have an output of 0. Hence if a database exists with the above 3 attributes, i.e. age, height and hair, with respectively 2, 3 and 4 different qualitative values, then the object description vector in the paragraph above would be coded as 1 0 0 1 0 1 0 0 (reading from left to right and assuming that 'young' is the first value of age, 'tall' is the third value of height, and 'brown' is the second value of hair). See FIG.3.

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>VALUES</th>
<th>NBR. INPUT NEURONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>young, old</td>
<td>2</td>
</tr>
<tr>
<td>height</td>
<td>short, medium, tall</td>
<td>3</td>
</tr>
<tr>
<td>hair</td>
<td>black, brown, blond, grey</td>
<td>4</td>
</tr>
</tbody>
</table>

Input vector  ((AGE = young) (HEIGHT = tall) (HAIR = brown))

FIG. 3 FORMATTING INPUT DATA

Once all the object description vectors have been converted to this binary format, they can be presented to the neural network input layer. In practice, the input vectors are presented in random order and a user specified number of times. A 1 in an input vector indicates that the output of the corresponding input-layer neuron has the value 1. A 0 in the input vector indicates that the corresponding input neuron is inactive, having an output of 0.
The complete data base of input vectors is presented a sufficient number of times for the weights connecting the input to the output neurons to saturate, i.e. they no longer change significantly in value. Weights are initialised randomly, such that the weights of each neuron sum to 1. The number of output neurons is user specified and will vary between 2 and the total number (V) of input vectors. The output nodes can be looked upon as cluster detectors, one cluster per output neuron. Since each output neuron detects a cluster, there can be no greater number of clusters than the number of input vectors. By varying the number of output neurons systematically from 2 to V (although in practice it is rare that one goes as far as having V output neurons), and presenting the data several times for each number of output nodes, a family of clusters can be detected. This detection occurs as follows.

If a cluster of objects has the same value for a given attribute, then the corresponding weight on the output neuron (corresponding to that cluster) will be "large", and the other weights (corresponding to the other values for the same attribute) will be small in comparison. For each attribute, the proportion of the weight corresponding to each value, relative to the total of the weights of the values of the attribute, are calculated. For example, if an attribute has 4 possible values, there will be 4 connections from the input neurons to the output neuron corresponding to the cluster concerned. If these 4 weights are w1, w2, w3, w4 respectively, then the proportion of w1 with respect to the total of the weights for the attribute is simply w1/(w1+w2+w3+w4). If this proportion is close to 1, then for the cluster of objects concerned, the attribute can be considered to take its first value. It is detected.

We have thus a means to discover clusters via their descriptions. The user specifies a "threshold detection value" (usually quite high, e.g. 90% or 95%) which is used to see whether a given attribute value has been detected with sufficient reliability. If not, that is, if none of the proportional weights is above threshold, then no value for the given attribute is detected. Such an attribute takes a "don't care" value (represented as "d" in this paper). By examining all the weights, the description of a cluster can be found, e.g. d3d3d4dd is the description of a cluster whose second attribute takes its third value, and whose sixth attribute takes its fourth value. The other attribute values are "don't cares".

The objects whose descriptions match a cluster description become the elements of that cluster. By looking at the weight proportions of each output neuron, where the number of output neurons varies from 2 to V, and by repeating the whole process (a user specified number of times) for the same number of output neurons, a family of such cluster descriptions and the objects which match these descriptions are found. See FIG. 6 in section 5 for an example of a list of cluster descriptions and cluster elements.

Hidden in this list are subsets of clusters which form partitions (or as close as possible to partitions) over the objects concerned. To find these subsets, the genetic algorithm is employed.

3. The Genetic Algorithm:

This section gives a very brief introduction to the basic principles of the genetic algorithm (GA). The GA is well explained elsewhere [GOLDBERG 1989]. The GA is essentially a "weak method" search procedure, used to optimize some process or function. It employs techniques which are analogous to Darwinian evolution, e.g. survival of the fittest, reproduction, chromosomes, mutation, crossover, etc. Coded "solutions" to a problem (e.g. parameter values) are placed on "chromosomes" (usually binary strings) which when decoded, specify the behaviour or value of a system which is to be optimized.
A population of such chromosomes compete with each other in transmitting (possibly modified) copies of themselves to the next generation. A quality measure of some kind is specified by the human user, which is applied to the "solutions" derived from the chromosomes. The chromosomes are ranked according to their quality values. Superior chromosomes reproduce more, inferior solutions reproduce less or not at all, such that the total population remains fixed.

So-called "genetic operators" such as mutation (e.g. bits switch from 0 to 1 or from 1 to 0), crossover (where two chromosomes swap the same portion of their strings), inversion (where a portion of a chromosome is inverted) are applied to the "offspring" chromosomes. The theory justifying these operators can be found in [HOLLAND 1975, GOLDBERG 1989].

With superior chromosomes being reproduced preferentially, and inferior chromosomes being eliminated from generation to generation, the average quality of the population improves, while the genetic operators ensure the creation of variety. The GA is a powerful technique to optimise a system with a large number of parameters. It was for this reason that it was chosen to find the subsets of clusters which form partitions over the object population.

4. Selecting Partition Clusters:

To use the GA, a suitable representation for the chromosomes needs to be chosen. COMPO uses the following format. Each cluster is given a numerical identifier (I.D.) (from 1 to M, where M is the number of clusters in the set). A random assortment of these identifiers is placed on a set of chromosomes (of a user specified length). For example, a third of the length of each chromosome is occupied by these chromosome I.D.s. The remaining slots on the chromosome are filled with zeros. Such a chromosome is interpreted as follows. The set of I.D.s on the chromosome is treated as a potential partition. FIG.4 shows a chromosome with 3 I.D.s. The object clusters corresponding to these 3 I.D.s are also depicted, as well as their superposition. Any unoccupied slot in the superposition is called a 'hole'. Any slot which contains more than 1 object is called an 'overlap'. With these definitions, the quality (Q) of a partition candidate (a chromosome) is measured according to the following formula:

\[ Q = \frac{\text{nbr of objects - nbr of holes -nbr of overlaps}}{\text{nbr of objects}} \]

Q takes the value 1 when a perfect partition is formed from the superposition. The aim is to use the GA to find a superposition with the highest possible Q value.

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>0 0 0 3 0 7 0 2 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 2</td>
<td>d1dd2dd7</td>
</tr>
<tr>
<td>Cluster 3</td>
<td>dd3ddddd</td>
</tr>
<tr>
<td>Cluster 7</td>
<td>ddd2dd3d</td>
</tr>
<tr>
<td>Superposition</td>
<td><strong>_</strong><em>_</em>**</td>
</tr>
<tr>
<td>Occupancy</td>
<td>2 1 0 2 1 0 2 2 1</td>
</tr>
</tbody>
</table>

Nbr of holes (occupancy = 0) = 2  Nbr of overlaps (occupancy > 1) = 5
Nbr of objects in set = 10  Q = (10 - 2 - 5)/(10) = 0.30

FIG. 4 PARTITION QUALITY
The GA used in COMPO is relatively straightforward and is taken largely from [GOLDBERG 1989]. Reproduction is based on the roulette wheel technique (i.e. the probability of a chromosome reproducing is proportional to its Q value divided by the sum of the Q values of the population). The genetic operators are crossover, inversion, and two forms of mutation. The Q values are scaled before reproduction to avoid premature convergence.

Crossover is implemented as follows. All chromosomes in the population are paired randomly and a user specified fraction of these pairs exchange a portion of their strings. Inversion is straightforward. Each chromosome is inverted (cut in two places and the dissected section inverted) with a user specified probability. Two kinds of mutation operators are used - one which writes a randomly chosen cluster I.D. at each chromosome slot (with a small user specified probability, e.g. 2%), and the second which writes a zero at each chromosome slot (with a small user-specified probability, e.g. 2%).

The first mutation operator is used to insert variety into the chromosome population. It is possible that cluster I.D.s are lost from the population as chromosomes are eliminated. The second mutation operator is used to remove I.D.s. The second operator can be used to cut down on the number of cluster I.D.s on a chromosome. The respective probabilities of application of these two mutation operators is user dependent, and can be used to bias the selection of partitions with few or many clusters.

To force the finding of a (near) partition with a given number of clusters (classes), the definition of the Q value can be modified. For example, if the number of clusters (number of I.D.s in the chromosome) is not equal to the number desired, then the above Q value can be halved. This strategy will obviously favour partition candidates with the desired number of clusters.

The Q values are scaled (linearly transformed) such that the average Q value of the population of the chromosomes is only a user-specified scaling factor times smaller than the maximum Q value (e.g. twice as small). This scaling reduces the probability of a premature convergence of the chromosome population to a local (non global) optimum Q value.

The GA used in COMPO is elitist, in the sense that the best chromosome (with the highest Q value in its generation) is automatically transmitted to the next generation. This strategy is used to avoid possible loss between generations of good partition candidates, considering that the roulette wheel reproduction strategy is stochastic.

5. A Simple Example:

To illustrate how COMPO works, a small data base of 30 members of the "cat" family is used, containing 14 attributes with qualitative values. FIG. 5 shows the data, where the 30 rows are the object vectors, and the 14 columns are the attributes. (A *2* means that the corresponding attribute takes its second value). FIG. 6 shows some of the clusters detected by the competitive learning portion of COMPO, as well as the two cluster descriptions (and their respective cluster members) which form a partition over the population of objects. Note that an asterisk means that the corresponding object matches the cluster description. An underscore means a no-match.

COMPO's GA found that clusters 5 and 21 gave a perfect partition (Q = 1), as shown in FIG. 6. The descriptions of clusters 5 and 21 are ((v3 = 2) & (v11 = 2)) and ((v11 = 1)) respectively.
| FIG. 5 DATA BASE FOR CAT FAMILY |

The clusters shown in FIG. 6 are a sample of a larger set. What is interesting in COMPO is that there appear to be rather few such clusters inherent in the data. This is a significant observation which results from many empirical trials, (e.g. changing the learning constant in the competitive learning, the weight detection threshold, the number of repetitions of the input vectors, the number of (random) reinitialisations of the weights, the number of output neurons, etc).

COMPO has several housekeeping modules used to store, delete and display cluster files, and to merge files containing these cluster descriptions and elements. When two such files are merged, repeated clusters (i.e. clusters possessing the same elements) are removed from the merge file. If two cluster descriptions match the same elements, then the cluster with the shorter description (fewer number of variables) is kept.
Using these housekeeping modules, one can incrementally build up a set of clusters (i.e. cluster descriptions and elements). One notices fewer and fewer additions to the accumulating set of clusters. Thus COMPO has significant advantages over traditional AI search techniques, where combinations of attribute-value pairs are tested to see if they match object descriptions. **COMPO finds the cluster descriptions directly**, so to speak. If two attributes often take the same two values simultaneously (i.e. they are correlated), then the two corresponding weights will be reinforced simultaneously.
COMPO as described so far produces partitions, not hierarchical classification trees. For the flat data above, the number of clusters in such partitions is usually small (2 or 3) and not necessarily "simple" (e.g. when an attribute may take two different values, and the description of one cluster (of two clusters in the partition) uses one value, and the other cluster uses the other value). Normally there are several partitions detected, in which case, further partition quality criteria need to be applied to choose between them, (e.g. using attributes which are cheap to measure, descriptions which are short, partitions which distribute their elements evenly over their clusters, etc.) There are many such additional user-defined partition quality criteria.

6. **Hierarchical COMPO**:

COMPO can be adapted to form hierarchical classification trees, by recursing the above two steps ("competitive learning" clustering, and "genetic algorithm" partitioning) at each node in the tree. FIG. 7 shows the first 2 levels of a classification tree (using data described in the next section). This recursion can be continued until the quality of a partition at a node falls below a user-defined threshold value. See section 7 for details.

COMPO has a module to extract clusters which match a given description (which can itself be extracted from a cluster file). At lower levels in the tree, one needs to know the elements contained in a given node in order to measure the partition quality of the subclusters which branch off from that node. It can happen (due to the stochastic nature of the algorithm) that a cluster description detected at a lower level, does not have the same value as its parent for the same attribute. If this occurs, such a cluster is simply eliminated from the cluster list, before the list is passed to the partitioning (GA) phase.

At lower levels in the tree, clusters may be hard to find. If there are too few clusters detected, then a partition at that node in the tree may have too poor a quality to be considered interesting. A user-specified partition quality threshold (e.g. Q = 90% or 95%) can be used to decide how deep to descend in the tree. If the Q value of a node does not pass threshold, that node is not expanded.

7. **A Larger Example**:

To see whether COMPO can handle larger, more real-world size data bases, Michalski’s Soybean data is used [MICHALSKI et al 1980]. This data consists of Soybean disease descriptions in the form of conjuncts of attribute-value pairs, using 50 attributes, with an average of 4 different possible qualitative values per attribute. The data consists of 17 classes of Soybean disease (as defined by a human plant pathologist), with 17 members per class. Once the input vectors are converted into binary form (194 input neurons), the data (17*17 = 289 input vectors) are presented a user-specified number of times.

Given that the Soybean data is already classified (by the plant pathologist), it is interesting to compare the clusters found by COMPO with these predefined classes. In FIG. 7 the elements are presented in blocks of 17 which correspond to the predefined classes. This format allows easy comparison between the "supervised" classes of Michalski and the "unsupervised" clusters of COMPO. Note that there are similarities between the two sets of results, but the correspondence is not perfect. Many of the predefined classes are split by COMPO. Note that L is the left node from the root, LR is the right subnode of L, etc.
FIG. 7 A "COMPO" CLASSIFICATION TREE
8. Conclusion and Future Research

The combination of competitive learning and the genetic algorithm in COMPO allows both intrinsic clusters to be detected, and then an optimal "partitioning" (hierarchical classification tree) over these clusters to be found. The great surprise of this work is the relatively restricted number of clusters detected, and with relatively little effort. The technique can be scaled up easily, and is thus an interesting alternative to traditional symbolic (artificial intelligence) clustering techniques which use "search" to form conjuncts of attribute-value pairs which match subsets of object descriptions. This computationally expensive technique is replaced by a more direct, data driven, connectionist approach. COMPO thus introduces connectionist techniques into Data Analysis, and into Conceptual (Descriptive) Clustering in particular. COMPO is said to be a Conceptual Clusterer in the sense that the clusters generated by the competitive learning have a description, and may be partitioned according to user-defined criteria.

Future research will need to compare the results of COMPO with other unsupervised learning classification techniques, such as Michalski's CLUSTER/2 [MICHALSKI et al 1984], Fisher's RUMMAGE [FISHER 1984], etc., using the same data bases and the same quality criteria. Further work also needs to be done in defining further partition (and perhaps classification tree) quality criteria. (See the LEF criteria in [MICHALSKI 1984], [BERGADANO et al 1988]). It will be easier if these criteria are applied at each node of the classification tree, because several alternative partitions at a node may be possible. This node by node application of partition quality criteria is simpler than building a combinatorially large number of trees and then measuring the quality of the whole tree with some form of classification tree quality criteria.

Acknowledgements :

This research was undertaken partially at the Artificial Intelligence Center of George Mason University, Virginia, U.S.A. The activities of the Center are supported in part by the Defence Advanced Research Projects Agency, administered by the Office of Naval Research, under grants Nos. N00014-87-K-0874, N00014-88-K-0226, and N00014-88-K-0397.

This paper also presents research results of the Belgian National Incentive Program for Fundamental Research in Artificial Intelligence, initiated by the Belgian State Prime Minister's Office, Science Policy Programming. The scientific responsibility is assumed by the author.

References :


