STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

EVOLUTIONARY CLUSTERING USING AN INCREMENTAL TECHNIQUE

R. GORUNESCU AND D. DUMITRESCU

Abstract. Since the various clustering methods developed over the time have failed to prove their flawless efficiency in the field, it might be that evolutionary computation holds the solution to this issue as well.

The goal of this paper is to present such an evolutionary technique with a classical clustering engine behind it.

Keywords: incremental clustering, evolutionary computation, genetic algorithms, merging, splitting, weighted similarity measures

1. Introduction

A new evolutionary clustering technique is proposed. This method represents an evolutionary variant of the incremental clustering technique.

Incremental clustering (IC)[6] is a powerful clustering method that is of great interest mainly because the number of clusters is not specified. This feature is very important in the field of unsupervised learning. Therefore, instances are added one by one forming a tree, starting with an empty root node. The best location for the new instance or the best restructuring of the part of the tree affected by it is determined by several operators, operators whose diversity leads us to the second reason behind the success of the IC method. The ordinary Euclidean distance - usually used for building an objective function - is replaced by a function called *category utility* which measures the quality of the partitioning.

2. Incremental Clustering

The idea behind the IC algorithm is quite simple. We start with an empty root. Instances are added sequentially until there are none remaining, as follows: for each instance we compute the category utility of placing it into an existing leaf versus the category utility of forming a leaf by itself. Whichever is better will determine the location of that instance. This is the general approach, but if we

Received by the editors: September 12, 2003.

2000 *Mathematics Subject Classification.* 62H30, 68T20.

1998 *CR Categories and Descriptors.* I.5.3 [Pattern Recognition]: Clustering – *Clustering algorithms*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search – *Heuristic methods*.

were to continue in this manner the resulting clustering would be dependent on the order in which instances were considered. Therefore, we move on to the next two operators that intervene now in the process. Firstly, there is the merging operator - that is combining two classes into a single one before the new instance is added to the resulting leaf - and secondly, the reverse one, that is the splitting operator - dividing a class into two. These two operators have proven to be extremely important in balancing the possible negative effects of the above mentioned ordering of instances.

2.1. ***Category utility*** **criterion.** A function to measure the quality of the clustering with a complex role is considered. The proposed function maximizes both the probability that instances in the same class have common attribute values and the probability that instances from different classes do not.

Let us consider the following notations:

$P(A = v|C)$ is the probability that an instance has value v for its attribute A, given that it belongs to class C. The higher the probability, the more likely instances in the same class will have attribute values in common.

$P(C|A = v)$ is the probability that an instance belongs to class C, given that it has value v for its attribute A. The higher the probability, the less likely instances from different classes will have common attribute values.

$P(A = v)$ is a weight of the fact that frequently occurring attribute values have a stronger influence on the evaluation.

Let $C_1, ..., C_k$ be the current partition. The category utility function $U(C_1, ..., C_k)$ is the quantity defined as follows:

$$U(C_1, ..., C_k) = \sum_C \sum_A \sum_v P(A = v)P(A = v|C)P(C|A = v),$$

where the first sum is taken with respect to all clusters, the second one with respect to all attributes of the members in class C and the third with respect to the attribute values.

However, this expression is not the one that is being used in practice. Instead, we will use a slightly changed form. This new expression is obtained by applying the Bayes formula for conditional probabilities, that is

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A),$$

and thus by simplying the first expression using the above formula, we obtain:

$$\sum_C \sum_A \sum_v P(A = v|C)^2 P(C)$$

The final form of the category utility will measure the amount by which information about what cluster the current instance is in does make a difference, compared to the situation of not knowing anything about the cluster structure, summed over all the clusters by their probabilities. Finally it is divided by the

number of clusters to discourage the phenomenon that each instance would be put in its own cluster.

3. A new evolutionary clustering algorithm

The drawbacks of the IC method are not particularly disturbing at large, with only one exception. To what extent is the final result dependent on the order of examples? Are the two operators - splitting and merging - sufficient to prevent this dependence?

The present method is desired to take care of this aspect through the techniques of evolutionary computation, on the one hand, and to take advantage of the characteristics of a powerful method, the incremental clustering, on the other hand.

The incremental nature of the IC method will be preserved precisely. The mechanisms of evolutionary algorithms will provide the several parallel possibilities of ordering the instances. The effect of the merging and splitting operations will be identical to that of the original method by means of the recombination and mutation operators. Finally, the expression of the fitness function will be inspired from the category utility criterion, but dwelt upon more from a similarity comparison between instances point of view rather than from a probabilistic one.

3.1. **Representation. Initial population.** The value of a gene will represent the cluster number of the instance labelled with the position of that gene. That is, if we denote by c the current chromosome, then c_i will give the number of the cluster in which the i-th instance will be, $i = 1, ..., m$, where m is the number of instances of the specified database. For example, if we have four objects, then the chromosome (1,3,3,2) means that instance one is in a cluster, instance four in another cluster, and instances two and three in the third cluster.

The initial population will be made up of chromosomes with a single 1-valued arbitrary position, that is for every chromosome we take randomly an instance and put it in the first cluster. In this way, the algorithm starts to offer several parallel possibilities of ordering the data, and continues furthermore in this sense by the means of a special variation operator.

3.2. **Fitness function.** First of all, we have to define the similarity measure between two instances of our considered database, since our function is built upon its expression. We have used a weighted similarity measure, since each attribute of our data has a different degree of importance in the field they are extracted from.

$$distance(a, b) = \sum_{k=1}^{n} compare(a_k, b_k),$$

where a and b are the two instances and n represents the number of attributes.

At this point there are two cases:

(i) If we are dealing with numerical attributes, the difference between the two attributes is the square weighted Euclidian similarity measure, that is:

$$compare(a_k, b_k) = (a_k - b_k)^2 weight_k,$$

where $weight_k$ is a positive number specifying the importance of attribute k.

(ii) In the other case, of the nominal attributes, we have considered their representation as fuzzy. Therefore, for the difference between such attributes, the max-min distance specific to fuzzy data is considered:

$$compare(a_k, b_k) = \max(\min_{i=1}^{n_k})(a_k^i, 1 - b_k^i) weight_k,$$

where n_k is the number of values for the k-th attribute of the chromosome.

For the expression of the fitness value, we will act as it follows.

Let c be the current chromosome and we would like to compute its performance. Then

$$eval(c) = \frac{\sum_{Clst=1}^{k} \sum_{i,j=1,\ldots,m, i<j, c_i=c_j=Clst} distance(instance_i, instance_j)}{k},$$

where $instance_i$ represents the i - th instance in the database and k represents the number of clusters denoted by that chromosome.

If a gene with a unique value in that chromosome is found, its penalty (instead of the *distance* function in the above formula) for forming a cluster of its own will be 1.

The division by the number of clusters prevents the phenomenon of too crowded clusters.

As it can be easily seen, this performance function is somewhat similar to the expression of the category utility in IC, with the significant difference that, while category utility has to be maximized, we are trying to minimize our fitness function.

Another fitness evaluation can be considered independent of the category utility criterion, but still considering the basic idea of clustering, that is minimizing intra-class distance and maximizing inter-class distance. Therefore, we are led to the following multi objective optimization problem (MOEA):

$$f_1(c) = \frac{\sum_k \sum_{c_i=c_j=c_k} d(instance_i, instance_j)}{|c_k|^2} \to min$$

and

$$f_2(c) = \frac{\sum_k \sum_{c_i=c_k, c_j \neq c_k} d(instance_i, instance_j)}{|c_k|^2} \to max$$

Standard MOEAs [1] can be used for solving our problem.

The output of a MOEA is a set of feasible solutions, the optimal Pareto solutions [1].

To avoid difficulty in choosing a single Pareto solution, we propose to combine the objective functions f_1 and f_2 in a unique criterion function:

$$F(c) = k_1 f_1 + k_2 \frac{1}{f_2}$$

We are led to $F \to min$.

3.3. **Variation Operators.**

3.3.1. *Recombination.* The standard 2:2 one point crossover operator is used. When used, it will produce either a merging or a splitting of the two clusters involved.

The best two individuals from both parents and offsprings are kept.

3.3.2. *Mutation.* As regarding mutation, special interest has to be paid, as two types are considered.

The first one is in charge of splitting. When a gene is considered for this kind of mutation, a second one that has the same value is searched for, and if a single one found, the current gene will get the number of the next cluster to be formed. Else, nothing happens.

The second type of mutation puts the current instance (given by the index of the current gene) in an existing cluster, whether that instance is or not part of a cluster containing other instances as well. This second mutation operator is clearly in charge either of splitting or merging.

Again the best individual among parent and offspring is accepted in the new population, in both cases.

3.3.3. *Increment.* A new variation operator — *increment* — is introduced. It is applied for every chromosome, taking randomly a gene of the current one, whose value is necessarily zero, and assigning it either the number for the next cluster to be formed or the number of an existing cluster. This operator is in charge of keeping the incremental nature taken over from the IC method. It actually puts an undistributed instance in a new or an existing cluster.

3.3.4. *Stop condition.* The algorithm stops when, after a number of iterations, considered equal in value to the number of the objects in the data set, no progress in the value of the overall fitness function can be observed.

The best chromosome from the final population will give the optimal clustering.

3.4. **Other parameter settings and experimental results.** Consider a fictional data set that describes the weather conditions for playing some unspecified game[6] given in Table 1.

id	**outlook**	**temperature**	**humidity**	**windy**
x_1	(sunny-0.78,overcast-0.45,rainy-0.20)	(hot-0.90,mild-0.50,cool-0.10)	(high-0.78,normal-0.12)	(true-0.13,false-0.90)
x_2	(sunny-0.80,overcast-0.34,rainy-0.10)	(hot-0.80,mild-0.40,cool-0.20)	(high-0.80,normal-0.20)	(true-0.89,false-0.23)
x_3	(sunny-0.30,overcast-0.85,rainy-0.34)	(hot-0.90,mild-0.30,cool-0.10)	(high-0.90,normal-0.30)	(true-0.16,false-0.77)
x_4	(sunny-0.10,overcast-0.50,rainy-0.90)	(hot-0.40,mild-0.80,cool-0.50)	(high-0.70,normal-0.10)	(true-0.22,false-0.86)
x_5	(sunny-0.13,overcast-0.50,rainy-0.70)	(hot-0.10,mild-0.50,cool-0.80)	(high-0.30,normal-0.80)	(true-0.15,false-0.88)
x_6	(sunny-0.20,overcast-0.40,rainy-0.87)	(hot-0.20,mild-0.40,cool-0.90)	(high-0.30,normal-0.79)	(true-0.77,false-0.30)
x_7	(sunny-0.50,overcast-0.80,rainy-0.30)	(hot-0.30,mild-0.20,cool-0.92)	(high-0.40,normal-0.98)	(true-0.89,false-0.20)
x_8	(sunny-0.90,overcast-0.70,rainy-0.10)	(hot-0.60,mild-0.80,cool-0.20)	(high-0.84,normal-0.22)	(true-0.14,false-0.88)
x_9	(sunny-0.78,overcast-0.34,rainy-0.20)	(hot-0.20,mild-0.60,cool-0.96)	(high-0.13,normal-0.95)	(true-0.10,false-0.98)
x_{10}	(sunny-0.10,overcast-0.50,rainy-0.70)	(hot-0.10,mild-0.90,cool-0.50)	(high-0.24,normal-0.87)	(true-0.34,false-0.68)
x_{11}	(sunny-0.80,overcast-0.30,rainy-0.10)	(hot-0.20,mild-0.87,cool-0.40)	(high-0.32,normal-0.89)	(true-0.56,false-0.45)
x_{12}	(sunny-0.40,overcast-0.90,rainy-0.30)	(hot-0.12,mild-0.90,cool-0.60)	(high-0.82,normal-0.30)	(true-0.85,false-0.30)
x_{13}	(sunny-0.20,overcast-0.90,rainy-0.50)	(hot-0.90,mild-0.50,cool-0.20)	(high-0.40,normal-0.80)	(true-0.65,false-0.22)
x_{14}	(sunny-0.10,overcast-0.30,rainy-0.90)	(hot-0.40,mild-0.78,cool-0.11)	(high-0.98,normal-0.14)	(true-0.94,false-0.12)
	0.2	0.3	0.1	0.4

TABLE 1. The weather data set

We consider the values for the other parameters involved given in Table 2.

population size	**recombination probability**	**mutation probability**
100	0.7	0.7

TABLE 2. Algorithm parameter values

The best chromosome obtained is:

5 5 3 6 2 7 7 6 4 2 4 1 8 1

Therefore, the corresponding classes are:

$$A_1 = \{x_1, x_2\},$$

$$A_2 = \{x_3\},$$

$$A_3 = \{x_4, x_8\},$$

$$A_4 = \{x_5, x_{10}\},$$

$$A_5 = \{x_6, x_7\},$$

$$A_6 = \{x_9, x_{11}\},$$

$$A_7 = \{x_{12}, x_{14}\}$$

and

$$A_8 = \{x_{13}\}.$$

What is very encouraging is that all the final chromosomes provide in 9 cases out of 10 the following clustering results:

(i) instances x_6, x_7 in a cluster,
(ii) instances x_5, x_{10} in a cluster,
(iii) instances x_3, x_8 in a cluster,
(iv) instances x_1, x_2 in a cluster, and
(v) instances x_{12}, x_{14} in a cluster.

3.5. **Conclusions and future work.** The proposed method provides a good enough clustering method. In future work, a better control over the merging and the splitting is desired.

Moreover, a comparison between the results of the original IC method and the evolutionary one would be of real interest.

References

[1] Coello Coello, C., Van Veldhuizen, D., Lamont, G., Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic/Plenum Publishers, New York, 2002.
[2] Dumitrescu, D., Genetic Algorithms and Evolution Strategies, Blue Publishing House, Cluj-Napoca 2000
[3] Dumitrescu, D., Lazzerini, B., Jain, L., C., Dumitrescu, A., Evolutionary Computation, CRC Press, Boca Raton, Florida, 2000
[4] Gorunescu, R., Evolutionary Incremental Clustering. A New Technique for Detecting Natural Grouping, Research Notes in Artificial Intelligence and Digital Communications, 103, 2003, 73–81
[5] Michalewicz, Z., Genetic Algorithms + Data Structures + Evolution Programs, 2nd edition, Springer - Verlag, 1992
[6] Witten, I., H., Frank, E., Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 1999

Faculty of Mathematics and Computer Science, Department of Computer Science, University of Craiova, 13 Al. I. Cuza 1100 Craiova Romania
E-mail address: ruxandragorunescu@yahoo.com

Faculty of Mathematics and Computer Science, Department of Computer Science, Babes-Bolyai University, 3400 Cluj - Napoca Romania
E-mail address: ddumitr@cs.ubbcluj.ro