A hybrid genetic algorithm / decision tree approach for coping with unbalanced classes.

Deborah R. Carvalho (deborah@ipnet.com.br)
Bráulio C. Ávila (braulio@ppgia.pucpr.br)
Alex A. Freitas (alex@ppgia.pucpr.br)

Pontifícia Universidade Católica do Paraná, PUC-PR
PPGIA - Programa de Pós-graduação em Informática Aplicada
Rua Imaculada Conceição, 1155 - Prado Velho
CEP 80215-901 - Curitiba, Paraná - BRASIL
phone coml: [011-55] 41-330-1654
fax: [011-55] 41-330-1669

Abstract

This paper proposes a new approach for coping with the problem of unbalanced classes, where some class(es) is(are) much less frequent than the other(s). The proposed approach is a hybrid genetic algorithm / decision tree system. The genetic algorithm acts as a wrapper, using the output of a decision tree algorithm (the state-of-the-art C5.0) to compute the fitness of population individuals (candidate solutions to the problem of unbalanced classes). We evaluate the proposed system on a case study application domain about census data.

Key words: unbalanced classes, decision tree, hybrid genetic algorithm, wrapper, C5.0

1 Introduction

This paper addresses the well-known task of classification, where the aim is to predict the value of a goal attribute given the values of several predicting attributes.

In our case the goal attribute represents an overall measure of the quality of life of a given city, and the predicting attributes represent several individual indicators about the infrastructure of that city. For instance, some predicting attributes are the percentage of houses with water supply, the percentage of houses with collected garbage, and so on. (These indicators have, of course, a formal definition, but this is beyond the scope of this paper.) There are 16 predicting attributes.

The values of the goal attribute (the classes) are defined by the System of Analytical Indicators of IPARDES (Parana State's Institute of Social and Economic Development) [4].

The data being mined was collected in the national census of 1991 (one census is performed every ten years in Brazil), performed by the IBGE (The Brazilian Institute of Geography and Statistics) [3], which is also the institute responsible for the formal definition of the predicting attributes used in our case study.

The data set being mined contains information about the infrastructure and quality of life of all 323 cities in Parana State. For each city there are two examples (cases) in the data set, one of them with data about the urban area of the city and the other one with data about the rural area of the city. (An exception is the city of Curitiba, the capital of Parana State, which has only an urban area.) Hence, the data set contains 645 examples.

The goal attribute, quality of life, can take on four distinct values (classes), with the following meaning and proportion in the data being mined (Table 1.1):

Table 1.1 Meaning and proportion of goal attribute's values (classes)

Class	Meaning	Proportion (%)
"one"	City in very critical condition	42
"two"	City in critical condition	46
"three"	City in reasonable condition	7
"four"	City in good condition	5

As we can see in Table 1.1, the class distribution is very unbalanced. Classes three and four occur with a frequency much smaller than classes one and two. Obviously, this renders the rule induction process significantly more difficulty, and this problem is the major challenge faced in this paper.

Hence, we propose a new method for coping with the problem of unbalanced classes, and evaluate the proposed method in a case study – the above–discussed application domain.

The proposed method is based on the idea of expanding the original training set by creating multiple duplicates of some existing examples, in order to create an expanded training set with a more evenly-balanced class distribution.

The crucial question in this approach is: how many times should each example be duplicated in the training set? The answer is not trivial, because there is a trade-off between two conflicting goals. On one hand, we would like the class distribution to be as even as possible. On the other hand, we would like to use a training set whose information contents is as close as possible to the original data (since the test set presumably follows the same data distribution as the original data set).

To answer this difficult question, we propose a genetic algorithm, as described in the next section.

This paper is organized as follows. Section 2 describes in detail our hybrid genetic algorithms / decision tree method. Section 3 presents the results of computational experiments applying our algorithm to our case study data set. Finally, section 4 concludes the paper.

2 Algorithm

The basic idea of our method for coping with unbalanced class distributions is to use a genetic algorithm to optimize the number of copies (duplicates) of each example in the training set.

The proposed method follows a wrapper approach, combining a genetic algorithm (GA) with a decision tree (DT) algorithm. A wrapper is an algorithm that treats another algorithm as a "black box" and uses the output of the latter in its processing [6]. In our case, the GA treats a DT algorithm as a black box. More precisely, the fitness function of the GA is based on the predictive accuracy achieved by the DT algorithm on an expanded training set (with duplicate examples). This approach is somewhat analogous to the approach discussed in [10]. The major difference is that [10] uses a GA to optimize attribute costs, whereas we use a GA to optimize the number of copies (duplicates) of each example in the training set.

The DT algorithm used in our work is C5.0, developed by Ross Quinlan from his previous algorithms ID3 and C4.5 [8] and [9]. This algorithm was chosen for two reasons.

First, it is a state-of-the-art data mining algorithm, with many improvements over previous DT and rule induction algorithms. Second, it was recently acquired by PUC-PR (Pontifical Catholic University of Parana at Curitiba) and is currently being the target of several tests in other case studies parallel to this one, at this university.

2.1 Individual Encoding

Each individual of the population consists of several features. Each feature contains a value representing how many times a given group of examples is to be replicated in the training set.

In our case study application domain, each individual consists of 56 features. This number was derived as follows. The two most important predicting attributes are the city's type of area, which can take on 2 values (urban or rural), and the city size, which can take 7 discrete values (varying from very small to very large sizes, according to number of inhabitants) [1]. In addition, there are four classes, as explained in the introduction.

Each combination of city's type of area, city size and class corresponds to a single group of examples for the GA. For each of these groups of examples a feature determines how many times every example in the group is replicated in the training set. Hence, there are 56 (2 x 7 x 4) features.

The value of each feature (i.e. how many times every example of a group is replicated in the training set) has well-defined limits. The lower limit is 1 and the higher limit is the ratio of the number of observations of the most frequent class over the number of observations of the least frequent class. An exception is that the value of a feature is 0 if there is no training example in the definition of the group of examples associated with the feature.

The Figure 2.1 shows an example of an individual's genotype (parameter set). This figure shows the value for each of 56 features of an individual. In this figure the value of first feature is 1. This means that every training example having <class = one, city's type of area = urban, city size = 1> will have just one copy (no replication) in the training set. In this figure the value of the second feature is 5. This means that every training example having <class = one, city's type of area = urban, city size = 2> will have five copies in the training set; and so on.

Figure 2.1 Example of an individual's genotype.

1 5 5 0 5 7 1 2 2 5 0 6 5 1 2 4 5 10 9 0 1 2 1 5 9 8 0 1 2 2 10 6 1 0 3 9 5 9 5 5 0 1 2 8 6 6 4 2
8 5 4 1 4 8 0 2

In the initial population (generation zero) the individuals are generated in a random fashion, as usual in GAs. In the future, it might be interesting to try some seed initialization method based on class distribution, as suggested by an anonymous reviewer. This would introduce problem-dependent information into the initial population, which could improve the quality of solution found by the GA.

To summarize, each individual specifies how many times each training example is to be replicated, so that an expanded training set is produced for each individual. The expanded training set is then given for C5.0, and the predictive accuracy achieved by this latter is used to compute the fitness of the individual, as explained in the next section.

2.2 Fitness Function

The available 645 examples were divided into 3 subsets, viz. a training (Tr) set, a validation (V) set and a test (Ts) set, with 387, 129 and 129 examples respectively. The same Tr and V sets were used in each generation of the GA. In Tr and V sets along the generations would influence the quality of the solution found by the GA, as suggested by an anonymous reviewer.

During the evolution of the GA, for each individual of the current population the Tr set is expanded according to the individual's genotype (as explained in the previous section). The expanded training set is then given to C5.0, which builds a classifier from that set. The constructed classifier's performance is evaluated on the V test. The result of this evaluation is used to compute the fitness of the individual, i.e. a measure of the quality of the solution associated with that individual. (The better the fitness value of an individual, the more its features will be passed on to the next generation.)

Note that the Ts set is used only after the evolution of GA is completed. Then the Ts set will be used to evaluate the predictive accuracy of C5.0 associated with the best training set expansion proposal (best individual) found by the GA.

We have performed two sets of experiments, whose results will be described in section 3. Each set of experiments used a different fitness function, as follows.

In the first set of experiments the fitness function was simply the error rate on unseen data output by the C5.0 software. For instance, in the example of Figure 2.2 the error rate is 36.4%. Note that what the C5.0 software calls "test" data is, in our case, the V set, as explained above.

Figure 2.2 Example of the output of C5.0

```
C5.0 INDUCTION SYSTEM [Release 1.06]  Thu Nov 12 11:13:20 1998
------------------------------------
    Options: File stem <sub1235>
Read 516 cases (16 attributes) from sub1235.data

Evaluation on test data (129 cases):
        Decision Tree
      ----------------
      Size      Errors
        55   47(36.4%)    <<

       (a)   (b)   (c)   (d)   <-classified as
      ----  ----  ----  ----
        41    13                (a): class um
        15    41           4    (b): class dois
         3     6                (c): class tres
         1     5                (d): class quatro
```

In the second set of experiments the fitness function was defined in a more elaborated fashion, by calculating an accuracy rate which explicit takes into an account the problem of unbalanced classes. We call this an accuracy rate sensitive to unbalanced classes, or simply accuracy rate for short. (Note carefully that this accuracy rate is not the dual of the error rate of the first experiment, since this latter does not explicit considers the problem of unbalanced classes.)

The accuracy rate used as fitness function in the second set of experiments is computed in two steps, as follows.

- First, compute the percentage of correct classifications for each class. In the example show in Figure 2.2, this percentage for the class one is 41/54 *100 = 75.9%, since there are 54 (41 + 13) examples of class one, out of which 41 are correctly classified by C5.0. The same process is repeated for each of the four classes.
- Second, the accuracy rate is computed by multiplying the percentage of correct classifications for each class and taking the 4-th root, as formally shown in formula (2.1) [7]. This formula effectively computes the geometric mean of the four percentages computed in the first step.

$$\text{Accuracy Rate} = ((A_1 * A_2 * A_3 * A_4) ** (1 / 4) \qquad (2.1)$$

where A_i , $i = 1, .. , 4$, is the percentage of correct classifications for the i-th class.

The reader might ask why we should use, as fitness function, such an elaborate accuracy rate, rather than the much simpler error rate output by C5.0. The answer is that the former is a fairer measure of the performance of a classifier when the data being mined suffers from the “curse” of unbalanced classes (which is the critical problem being addressed in this paper). To see that this is true, consider, for instance, an application domain where the most frequent class occurs in 99% of the examples. In this case we can trivially achieve an error rate of just 1%, by simply always predicting the most frequent class. However, this would obviously be an undesirable classifier, which learned nothing (except the class distribution) from the data.

In the example of Figure 2.2 this trivial classifier would have an accuracy rate (sensitive to unbalanced classes) of 0 (zero) – correctly indicating the bad quality of the classifier, since the classifier did not correctly classify any example belonging to classes three and four.

2.3 Genetic Operators

To evolve a population of individuals we need to apply genetic operators that produce new individuals (which will form the next generation) from the best individuals of the current generation.

We briefly describe below the genetic operators used by our GA. If the reader is not familiar with these operators, he/she is referred to [2] or [5].

We used a population size of 100 individuals – a parameter value often used in the literature and adequate for our experiments.

The reproduction operator was tournament selection with elitism. We used a tournament size of 2, i.e. two individuals are randomly selected and the best one (the one with better fitness value) is selected for reproduction. Out of the 100 individuals, 98 are reproduced by tournament selection and the best 2 individuals of the current population are passed on, unaltered, to the next generation, in an elitism-reproduction scheme. Elitism avoids the danger of possible loss of the best two individuals of the current generation. Note that this loss might occur if tournament selection was used alone (without elitism), due to the stochastic nature of tournament selection.

The individuals selected for reproduction undergo the application of two other stochastic operators, namely mutation and crossover. In our case study mutation is applied with a 1% probability and crossover with a 80% probability. We used the conventional 1-point crossover. The mutation operator replaces the current value of a feature with a randomly generated, valid value. By valid we mean the new feature value must be in the range 1 ... k, where k is as defined in Section 2.1.

In all the experiments reported in this paper, each run of the GA consisted of 100 generations. This is a parameter value often used in the literature and was adequate for our experiments, as will be seen in the next section.

3 Computational Results

In order to determine whether or not our wrapper approach improves the results achieved with C5.0 alone trained on the original (badly-balanced) data set, first of all we have performed the following procedure.

We built 3 classifiers, by running 3 versions of C5.0, namely "default" C5.0, C5.0 with boosting and C5.0 transforming the decision tress into a set of rules (C5.0 with rules, for short). These classifiers were built from a training set formed by the sets Tr and V (see section 2.2), and were tested over the set Ts.

For each test performed, we retrieved the error rate output by the C5.0 software and calculated the accuracy rate sensitive to unbalanced classes – as defined by formula (2.1). The results are reported in Table 3.1

Table 3.1 – Results of 3 versions of C5.0 on the original (unexpanded) training base.

	acuracy rate	error rate output by C5.0
C5.0 default	31.35	50.40
C5.0 boosting	0.00	32.60
C5.0 rules	0.00	46.50

As shown in the second column of Table 3.1, only default C5.0 achieved an accuracy rate (sensitive to unbalanced classes) greater than 0 (zero). This means that C5.0 with boosting and C5.0 with rules failed in correctly classifying at least one example of the rarer classes (classes three and four).

On the other hand, as shown in the third column of Table 3.1, default C5.0 achieved the worst error rate between the three versions of C5.0, with the best error rate being achieved by C5.0 boosting.

We now report the results for two sets of experiments using our approach. The first one uses the error rate fitness function and the second one uses the accuracy rate (sensitive to unbalanced classes) fitness functions (as discussed in section 2.2).

3.1 Experiments with the error rate fitness function

These experiments consisted of 6 runs of the GA, with different initial populations (varying the random seed of the GA). For each of the runs of the GA, once the evolution was completed the sets Tr and V were merged and then expanded, i.e., the merged training set had its examples replicated according to the example distribution associated with the genotype of the best individual found by the GA.

For each of these 6 expanded training sets, 3 classifiers were built – default C5.0, C5.0 with boosting and C5.0 with rules. (Hence, in total 18 classifiers were built).

Table 3.2 shows the results of the 6 runs of the GA. The second column of this table shows the total number of examples in the expanded training set associated with the best individual found by each GA run. The third column shows the number of the generation were the population "stabilized" – i.e., all individuals converged for the same genotype. One can see that a number of generations larger than 100 would not significantly alter the results, since the stabilization of the population occurred, on average, around the 59th generation.

Table 3.2 Results of 5 GA runs with error rate fitness

	# examples	stabilization generation
run 1	1469	99
run 2	1737	3
run 3	1826	1
run 4	1761	78
run 5	1681	92
run 6	1679	85
average	1692.17	59.67

Table 3.3 shows the error rate figures achieved by the 3 versions of C5.0 when each version was trained on the expanded training set associated with the best individual of each of the 6 GA runs.

The first column of Table 3.3 identifies the version of C5.0. The second column of this table indicates the error rate of each version of C5.0 (alone) trained on the original (unexpanded) training set. This column is in reality the last column of Table 3.1, which is copied into Table 3.3 for convenience of the reader. The next 6 columns show the results of our hybrid genetic algorithm / decision tree approach. More precisely, the next 6 columns of Table 3.3 indicate the error rate for each version of C5.0 trained on the expanded training set associated with the best individual of the GA run indicated by the corresponding column's heading. The 9th column of Table 3.3 shows the average rate over the previous 6 columns. Finally, the last column of Table 3.3 shows the % of variation between the 9th and 2nd column of the table. As can be seen in the last column, our hybrid wrapper approach was most helpful for C5.0 default, where our approach reduced in 18.22% the error rate associated with C5.0 default alone. Also with C5.0 rules our wrapper approach was helpful, reducing 10.25% the error rate. But with C5.0 boosting it was not the same, our wrapper approach increased 1.48% the error rate. Maybe this little increasing occurred because the boosting algorithm have been developed for solving the problem with the more problematic examples, in our domain, the unbalanced frequency among the classes. What means that boosting is a very well optimized algorithm

Table 3.3 Error rates of C5.0 for the expanded training set associated with each GA run's best individual

	original training set	run 1	run 2	run 3	run 4	run 5	run 6	average	Variation (%)
C5.0 default	50.40	46.50	38.80	38.80	40.30	43.40	39.50	41.22	-18.22
C5.0 boosting	32.60	32.60	31.80	33.30	32.60	34.90	33.30	33.08	1.48
C5.0 rules	46.50	45.00	39.50	45.00	35.70	39.50	45.70	41.73	-10.25

3.2 Experiments with the Accuracy Rate (Sensitive to Unbalanced Classes) fitness function

These experiments also consisted of 6 runs of the GA, with different initial population (varying the random seed of the GA). For each of the runs of the GA, once the evolution was completed the sets Tr and V were merged and then expanded – i.e. the merged training set had its examples replicated according to the example distribution associated with the genotype of the best individual found by the GA.

For each of these 6 expanded training sets, 3 classifiers were built – default C5.0, C5.0 with boosting and C5.0 with rules. (Hence, in total 18 classifiers were built).

Table 3.4 shows the results of the 6 runs of the GA. The second column of this table shows the total number of examples in the expanded training set associated with the best individual found by each GA run. The third column shows the number of the generation were the population stabilized. Similarly to what was observed in Table 3.2, a number of generations larger than 100 would not significantly alter the results, since the stabilization of the population occurred, on the average, around the 36th generation.

Table 3.4 Results of 8 GA runs with accuracy rate fitness

	# examples	stabilization generation
run 1	1706	27
run 2	1769	84
run 3	1592	44
run 4	1727	2
run 5	1797	46
run 6	1761	17
average	1725.33	36.67

Table 3.5 shows the accuracy rate figures achieved by the 3 versions of C5.0 when each version was trained on the expanded training set associated with the best individual of each of the 6 GA runs.

The first column of Table 3.5 identifies the version of C5.0. The second column of this table indicates the accuracy rate of each version of C5.0 trained from the original (unexpanded) training set. This column is in reality the second column of Table 3.1, which is copied into

Table 3.5 for convenience of the reader. The next 6 columns show the results of our hybrid genetic algorithm / decision tree approach. More precisely, the next 6 columns of Table 3.5 indicate the accuracy rate for each version of C5.0 trained on the expanded training set associated with the best individual of the run indicated by the corresponding column's heading. Finally, the last column of Table 3.5 shows the average accuracy rate (sensitive to unbalanced classes) over the previous 6 columns.

As can be seen in the last column of table 3.5, our hybrid wrapper approach was significantly beneficial for C5.0 boosting and C5.0 rules. The accuracy rate of both versions alone was 0%, whereas our wrapper approach increased these to 31.34% and 37.00%, respectively. In run 6 our wrapper approach achieved an accuracy rate of 39.82%, against 31.35% of C5.0 default alone.

Table 3.5 Accuracy Rates (sensitive to unbalanced classes) of C5.0 for the expanded training set associated with each GA run's best individual.

	original training set	run 1	run 2	run 3	run 4	run 5	run 6	average
C5.0 default	31.35	34.11	36.70	44.20	47.58	46.48	29.86	39.82
C5.0 boosting	0.00	37.47	30.88	39.29	44.29	36.08	0.00	31.34
C5.0 rules	0.00	38.50	33.13	34.28	42.90	43.31	29.86	37.00

4 Conclusion

Overall, the expansion of the training set (i.e. a replication of training examples) achieved by using our wrapper approach can be considered a promising solution for the problem of unbalanced classes. Our computational experiments have shown that our wrapper approach – a hybrid genetic algorithm / C5.0 approach – obtains results which are, overall, significantly better than the results obtained by C5.0 alone applied to the original (unexpanded) training set.

A disadvantage of our hybrid GA / C5.0 approach is that it is much computationally shower than the use of C5.0 alone. However, for relatively small data sets the time taken by our approach is quite acceptable. In our case study application domain this processing time is on the order of 12 hours for each GA run[1].

It should be noted that in our case study application domain the data is very static, since new data is collected only when there is a new census (every ten years). Therefore, it is perfectly acceptable to spend several days of processing time with a data mining method, in order to improve the predictive accuracy of the discovered rules.

Finally, although this paper has focused on a case study about census data, we believe that our hybrid GA / C5.0 method is generic enough to be effectively applicable to other data sets. In any case, applying our method to other data sets suffering the "curse of unbalanced classes" is a necessary research direction, to better validate the effectiveness of the proposed method. Finally, another interesting research direction would be to use the high-level idea of our method but to replace C5.0 with another kind of data mining algorithm.

---

[1] Machine : Ultra Spark with 256Mb of main memory, running Unix
C5.0 release 1.6

Bibliographical references

[1] Andrade, Thompson A.; Serra, Rodrigo Valente, **O Recente Desempenho das Cidades Médias no Crescimento Populacional Urbano Brasileiro ("Recent Performance of Medium-Cities in the Brazilian Urban Population Growth")**, IPEA-Instituto de Pesquisa Econômica Aplicada, Rio de Janeiro. Março de 1998.

[2] Golberg, David, **Genetic Algorithms in Search Optimization, and Machine Learning.** Addison – Wesley, 1989.

[3] IBGE, **Censor Demographic Paranoia ("Demographic Census of Paranoia"),** Instituto Brasileiro de Geografia Estatística, Rio de Janeiro. 1992

[4] IPARDES, **Indicadores Analíticos ("Analytic Indicators"),** Volume I, Referencial Urbano, Curitiba. 1993.

[5] Michalewicz, Z. **Genetic Algorithms + Data Structures = Evolution Programs.** 3rd Ed. Springer-Verlag, 1996.

[6] Kohavi, Ron; Sommerfield, Dan; Dougherty, James, **Data Mining using *MLC*++ A Machine Learning Library in C++.** Appeared in Tool With AI 1996. http://www.sgi.com/Technology/mlc 1996.

[7] Kononenko, Igor; Bratko, Ivan, **Information-Based Evaluation Criterion for Classifier's Performance,** Machine Learning (6). 1991.

[8] Quinlan, J. Ross, **C4.5 Programs for Machine Learning,** Morgan Kaufmann Publishers, San Diego California, 1993.

[9] Quinlan, J. Ross, **Is C5.0 Better Than C4.5?,** 1997, htt://207.153.200.79/see5-comparison.html at 05/12/1997

[10] Turney, Peter D, **Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm,** Journal of Artificial Intelligence Research 2, Ottawa, Canada. March,1995