A Genetic Instance-Based Collaborative Approach for Attribute Weighting

Luciana De Nardin
Pontifícia Universidade Católica de Minas Gerais, Dept. de Ciência da Computação,
Poços de Caldas, Brasil, 37701-355
luciana@pucpcaldas.br

Maria do Carmo Nicoletti
Universidade Federal de São Carlos, Dept. de Ciência da Computação,
São Carlos, Brasil, 13565-905
carmo@dc.ufscar.br

Abstract

This paper shows that genetic algorithms can be used as an optimization tool in conjunction with an instance-based learning method, to produce a combination which improves the performance the learning method could achieve on its own. Two instance-based methods are investigated in collaboration with genetic algorithms, namely k-NN and IB2. We conducted a few experiments using a genetic algorithm for finding a 'good' weight vector for either learning algorithms. Classification results on three knowledge domains obtained using k-NN and IB2 modified by a weight vector found by a genetic algorithm, exceeds the performance of the instance-based methods on their own.

Keywords: instance-based methods, lazy learning, genetic instance-based collaboration, weighted NN, weighted IB2.

1. Introduction

Machine Learning is an area of research that provides a vast variety of learning models, algorithms, theoretical results and applications. Lately, a tendency in the development of new learning strategies based on the combination of well-established algorithms can be noticed in the area; the idea is that two learning algorithms can work together to outperform either individually. Among the many possible collaboration methods, one that seems promising is the use of genetic algorithms articulated to instance-based algorithms.

This work focuses on two instance-based algorithms namely, k-NN and IB2 and it is about the use of a genetic algorithm as an optimization tool for finding a weight vector to be used either by k-NN or IB2 aiming at improving their performance when classifying new examples. This paper is organized along the following lines. In Section 2, a general review of instance-based methods and particularly the two algorithms is intended. In Section 3, we describe a few characteristics of genetic algorithms that are relevant to this work and describe how the genetic instance-based collaboration was implemented focusing mainly on the fitness function. In Section 4 we present the main characteristics of the knowledge domains used in the experiments and discuss the results obtained from the collaboration. In the Conclusion, we list the next steps for continuing this work.

2. Instance-Based Learning – Considerations About the Algorithms K-NN and IB2

In contrast to methods that, based on training examples, construct a general description of the concept, instance-based learning methods simply store the training examples. Learning consists of storing the training examples in memory and never changing them. The concept description consists of the training set itself. For classifying a new instance, a distance (possibly weighted) is calculated between the new example and each stored training example and the new example is assigned the class of the nearest neighboring example. A generalization of this procedure takes into consideration the k nearest neighbors and the new example is assigned the class that is most frequent among these k neighbors [8]. The learning phase of these methods consists uniquely of storing; processing happens during classification time.

As commented in [14 – pg. 230] "instance-based methods are sometimes referred to as 'lazy' learning methods because they delay processing until a new instance is classified. A key advantage of this kind of delayed, or lazy, learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified."

The nearest neighbor algorithm (NN) [7] is the basis of many lazy learning algorithms. Basically NN techniques assume as the class of an instance x the class of the nearest instance from x. In order to determine the nearest instance, NN techniques adopt a distance metric that measures the proximity of instance x to all stored instances. Figure 1 presents the formal definition of the NN technique found in [9].

It assumes:

- n-dimensional feature space.
- M classes, numbered 1,2,…,M.
- p training instances, each one expressed as a pair (x_i, θ_i), for $1 \leq i \leq p$ where

a) x_i: training instance, expressed by a vector of pairs *attribute-value* $x_i = (x_{i_1}, x_{i_2}, ..., x_{i_{1n}})$

b) $\theta_i \in \{1,2,\ldots,M\}$ expresses the correct class of the instance x_i

Let $T_{NN} = \{(x_1, \theta_1), (x_2, \theta_2), \ldots, (x_p, \theta_p)\}$ be the nearest neighbor training set. Given an unknown instance x, the decision rule is to decide x is in class θ_j if

$$d(x,x_j) \leq d(x,x_i), \text{ for } 1 \leq i \leq p$$

where d is some n-dimensional distance metric.

Figure 1. 1-NN Algorithm

The algorithm described in Figure 1 is more properly called the 1-NN algorithm since it uses only one nearest neighbor. As mentioned earlier, one of the variants of the 1-NN algorithm is the k-NN algorithm, which takes into consideration the k nearest instances $\{i_1, i_2, \ldots, i_k\}$ and decides upon the most frequent class in the set $\{\theta_{i_1},$

θ_{i_2}, …, θ_{i_k}}. Algorithms derived from the nearest neighbor are very popular, mainly due to their simplicity, easy implementation and efficient results.

The k-NN algorithm treats all attributes in a similar way i.e., all the attributes are equally significant. There are situations, however, where the number of significant attributes (significant to the classification process) is small compared to the number of irrelevant attributes. When this happens the large number of irrelevant attributes will dominate the distance between neighbors and they will overcome the truly important attributes. As commented in [14, pg. 231], "…they (instance-based algorithms) typically consider all attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most 'similar' may well be a large distance apart."

Associating weights to attributes is a possible way to stress the relevance (or not) of attributes in the expression of the concept. A k-NN algorithm that implements a weight mechanism is generally referred to as Wk-NN.

Instance-based learning methods suffer from several problems and their main disadvantages are related to classification time and memory space, which are proportional to the number of stored examples. The two most relevant decisions to be made concerning these methods are: which training instances should be stored and which distance metric should be adopted in the classification phase, in order to 'measure' the distance of a new example to the stored instances that represent the concept.

Aiming at exploring the limits of instance-based methods, Aha et. al. proposed the IBL (Instance-based Learning) family of algorithms in [1] , which is strongly based on the nearest neighbor algorithm. IBL family groups five algorithms (IB1, IB2, IB3, IB4 and IB5). The first member of the IBL family is IB1 which can be considered the 1-NN algorithm renamed.

Because IB1 stores all training examples and each prediction of a new example involves calculating its distance to each of the stored examples, it becomes very inefficient when the training set becomes large. IB1's storage requirement however, can be reduced without decreasing too much its prediction accuracy by using a storage reduction algorithm from the IBL family, the IB2, which is used in this work. IB2 pseudo code is described in Figure 2.

```
CD ← ∅
for each x ∈ training set do
  begin
   for each y ∈ CD do
     sim[y] ← similarity(x,y)
     y_max ← some y ∈ CD with maximal sim[y]
      if class(x) = class(y_max)
         then classification ← correct
         else begin
                 classification ← incorrect
                 CD ← CD ∪ {x}
              end
  end
```

Figure 2. The IB2 algorithm (CD – Concept Description)

IB2 is identical to IB1 except that it only saves misclassified examples. As commented in [2], "The intuition in IB2's design is that the vast majority of misclassified instances are near-boundary instances that are located in the ε-neighborhood and outside the ε-core of the target concept (for some reasonably small ε)." In spite of IB2 storage reduction capabilities, this algorithm is much more sensitive to the presence of noise in the training set. This sensitivity to noise is a consequence of the fact that during learning, this algorithm only adds to the concept description the training examples that are incorrectly classified. Generally, noisy examples are incorrectly classified and consequently, they tend to be included in the concept description.

This paper is about combining both, k-NN and IB2 with a genetic algorithm aiming at improving the performance of either learning algorithm individually, by means of finding a suitable weight vector, which reflects the real contribution of each attribute that describes the concept. The GA will be used as a procedure that will carry out a search throughout an n-dimensional weight space 'looking for' suitable attribute weight vectors. The goal is to obtain a weight vector such that Wk-NN outperforms k-NN (and correspondently, the weighted version W-IB2 outperforms IB2).

3. Finding a 'Good' Weight Vector – A Contribution Given by a Genetic Algorithm

Although a k-NN which implements a weight strategy tends to have better performance than that which does not, it is very difficult to find a good weight vector. There are a few ways to define a weight vector associated to attributes. The user can define it, based on his/her experience on the knowledge domain. Another possibility is to conduct an exhaustive search throughout the space of all possible weight vectors, trying them all. Depending on the dimension of this space, such a search can be computationally unfeasible. A third option is to use a mathematical tool that could obtain, if not the best, at least a good weight vector which would improve the k-NN (or IB2) performance.

Finding a weight vector can be approached as an optimization problem which can be considered relatively difficult depending on the dimensions of the space to be searched. Several knowledge domains are described by as many as fifty attributes. The problem, in this situation, corresponds to a search for a vector in a 50-dimensional space, where the weight associated to each of the attributes is a real number.

A genetic algorithm (GA) is an adaptive general-purpose search algorithm, which has successfully been applied to many different problems in various areas. The basic principles of GA have been rigorously established by Holland in [12] and can be found in many references (see for instance [3], [4], [10] and [13]).

In GA the term population is used for naming a set of potential solutions to the problem; each individual solution is called a chromosome. Each part of a chromosome (usually representing a variable of the problem) is called a gene. Generally, the initial population is initialized with a pre-defined number of chromosomes which are randomly created; usually the number of individuals per population remains constant during the whole process. Inspired by the biological natural selection process, the GA through selection operator chooses the chromosomes from the current population in order to determine which individual candidates will be part of the 'reproduction' process.

As commented in [6], "Selection attempts to apply pressure upon the population in a manner similar to that of natural selection found in biological systems. Poorer performing individuals are weeded out and better performing, or fitter, individuals have a greater than average chance of promoting the information they contain within the next generation. Crossover allows solutions to exchange information in a way similar to that used by a natural organism undergoing sexual reproduction. Mutation is used to randomly change (flip) the value of single bits within individual strings." The process of selection, crossover and mutation goes on until a convergence criterion has been satisfied. Although there are many different variations of GAs, there is a canonical version, described in Figure 3.

```
procedure GA
begin
  t ← 0
  initialize(p(t))
  evaluate(p(t))
  while not (termination_condition) do
   begin
      t ← t + 1
      select p(t) from p(t-1)
      crossover(p(t))
      mutation(p(t))
      evaluate(p(t))
   end
end
```

Figure 3. Canonical GA

For the problem at hand, the initialization process consists of randomly creating a population of chromosomes, each of them representing a weight vector candidate to be the solution. The dimension of the chromosome is the number of attributes in the domain being considered. In a domain described by N attributes, each chromosome is a vector of N positions, each of them represented by a real number in the interval [0,1]. If the population has been established with size M, then M of such N-dimensional real vectors are randomly created.

The evaluation process uses either (k-NN or IB2) as the fitness function that 'measures' the quality of each chromosome in the population. In order to do that, a 10-fold cross-validation process was implemented; the fitness value of each chromosome is for the average values obtained using the ten learning-testing processes, as shown in Figure 4 and described as a pseudo code in Figure 5.

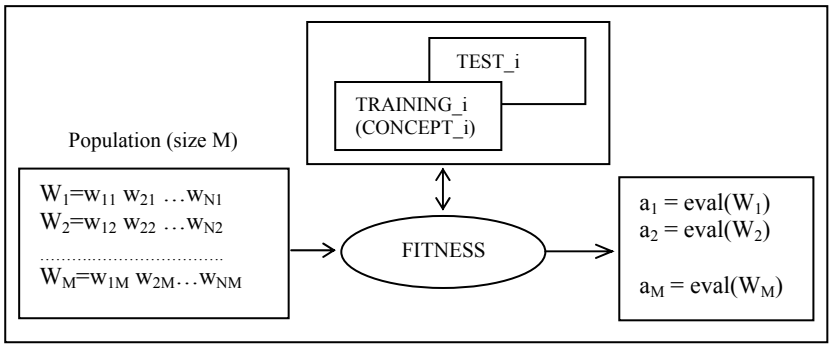

Figure 4. Using the accuracy of a learning algorithm as the fitness function. Each W_i ($1 \le i \le N$) is a weight vector and a_i its corresponding fitness value

The stopping criteria used in the experiments described in the next section was the number of generations; the value of k for implementing the k-NN was 5 (number of neighbors taken into consideration when classifying new examples). The crossover operator implemented is the one-point crossover, which is one of the simplest crossover operators and mutation was implemented as the random operator, which consists in substituting a gene by a random value from its domain.

```
procedure evaluate(P,C_K,T_K);
{P: population (size M) to be evaluated. Chromosome W_I ∈ P (1 ≤ I ≤ M) is a vector
   w_1I w_2I … w_NI , where N is the number of attributes in the domain
 C_K – concept description learnt from training set training_k (1 ≤ k ≤ 10)
 T_K – testing set (corresponding to training_k) to be classified by C_K, taking
       into consideration each W_I ∈ P.
 Each t_p ∈ T_K is a vector described as: t_1p, t_2p, t_3p … t_Np, class(t_p)        }

begin
 for_all W_I ∈ P do
  begin
   number_correct_classif(W_I) ← 0
   for_all t_p = (t_1p t_2p t_3p … t_Np) ∈ T_K do
    begin
     weighting(W_I,t_p,W_It_p)
     classify(C_K,W_It_p,R)
     if R then number_correct_classif(W_I) ← number_correct_classif(W_I) + 1
    end
    aval(W_I) ← number_correct_classif(W_I)/|T_K|
  end
end

weighting(W_I,t_p,W_It_p)
begin
 for q ← 1 to N do W_It_p[q] ← W_I[q]* t_p[q]
end

classify(C_K,W_It_p,R)
begin
 R ← false
 k_NN(C_K,W_It_p,Class)        {classifying with k-NN using a weight}
 if Class = class(W_It_p) then R ← true
end
```

Figure 5. Pseudo code of the evaluation process of a population using k-NN

4. Experiments and Results

The experiments conducted and described in this work are based on data from three knowledge domains, all of them with real-valued attributes. The domains Iris and Wine are well known domains and have been downloaded from the UCI Repository [5] – their descriptions can also be downloaded from the same site. The third domain named Vestibular is a dataset with the results of fixed saccadic tests performed on patients who attended the Service of Otoneurology of the Clinical Hospital which is part of the Medical School of the University of São Paulo, in Ribeirão Preto. The main characteristics of the three domains are shown in Table 1.

The Vestibular System domain data was provided by a medical researcher. Each example is a record for one patient. The data represents the measurement data collected by electrodes which were placed next to the patient's left and right eyes. The movements of both eyes were monitored as they focused on a spotlight, that shone alternatively from one extremity to the other of a horizontal bar, at a constant frequency, during a certain period of time. The electrodes measured the electrical signals which were produced by the saccadic movements. These signals were amplified, filtered and recorded for further analysis. The goal of physicians using these measurements is to be able to detect problems with the Vestibular System of a patient. More information about this domain can be found in [15] and [17].

Table 1. Main characteristics of the domains

Domain	Total Number of Examples	Total Number of Training Examples	Total Number of Testing Examples	Number of Attributes	Number of Classes	Number of Examples per Class
Iris	150	120	30	4	3	50 (setosa) 50 (virginica) 50 (versicolor)
Wine	178	143	35	13	3	59 (Region 1) 71 (Region 2) 48 (Region 3)
Vestibular	199	159	40	6	2	98 (Normal) 101 (Abnormal)

4.1 The Collaboration GA and K-NN in Order to Obtain a WK-NN

In the experiments conducted, we varied the size of the population (50 and 100 individuals) and the number of generations (20, 50, 100 and 500) in order to search for a set of genetic characteristics which would have the best performance. For all the experiments, we used the roulette selection operator, crossover rate of 0.8 and mutation rate of 0.01. The results shown in the next tables are for the best weight vector obtained in the last generation, considering the four different numbers of generations tried.

Iris Domain

Increasing the size of the population did not affect the results; the processing time, however, significantly increased. The significant performance of the population occurred between the 20^{th} and 40^{th} generation; the results of the Wk-NN using GA (obtained using a population size of 50 and generations number of 50) and the k-NN are shown in Table 2.

Table 2. Performance of k-NN versus Wk-NN using GA (Iris domain)

	k-NN	Wk-NN
Correct Classifications (%)	95.94	98.00
Standard deviation value	0.8498364548	0.0005900055

Vestibular Domain

The results in Table 3 showing the performances of both, k-NN and Wk-NN using GA are for size of population 50 and number of generations 50.

Table 3. Performance of k-NN versus Wk-NN using GA (Vestibular domain)

	k-NN	Wk-NN
Correct Classifications (%)	87.00	87.35
Standard deviation value	1.5491933384	0.0001399646

Wine Domain

Table 4 shows the performance values for the k-NN and Wk-NN using GA in the Wine domain, using a population and generation size each of 50. As can be seen in the table, the performance of Wk-NN is considerably inferior to that of the k-NN. In order to explore this domain more, we decided to eliminate the attributes considered less relevant because there was a chance of them interfering negatively in the search process. As suggested in [16], the fifth, sixth, eight and ninth attributes do not contribute much for characterizing the three classes in this domain. Based on this information, we reduced the domain to the nine attributes left and ran the experiments again. As can be seen in Table 5, both algorithms had their performances increased in the reduced domain; the improvement of the Wk-NN, however, was considerably higher compared to its performance on the complete domain.

Table 4. Performance of k-NN versus Wk-NN using GA (Wine domain)

	k-NN	**Wk-NN**
Correct Classifications (%)	78.88	71.77
Standard deviation value	1.3165610506	0.0032039615

Table 5. Performance of k-NN versus Wk-NN using GA (Reduced Wine domain)

	k-NN	**Wk-NN**
Correct Classifications (%)	90.00	90.35
Standard deviation value	1.2292725491	0.0012612338

4.2 The Collaboration GA and IB2 for Obtaining a W-IB2

We adopted the same procedure as described in the previous section i.e., in the experiments conducted, we varied the size of the population (50 and 100 individuals) and the number of generations (20, 50, 100 and 500) in order to search for a set of genetic characteristics which would have the best performance. For the experiments, we used the roulette selection operator, crossover rate of 0.8 and mutation rate of 0.01. The results shown in the next tables are for the best weight vector in the last generation, considering the four different numbers of generations tried.

Iris and Vestibular Domains

The results of the W-IB2 using GA (obtained using size of population 100 and 100 generations) and the IB2 are shown in Table 6 for Iris and Vestibular domains. Table 7 shows the best weight vector in the 100th generation for the Vestibular domain.

Table 6. Performance of IB2 versus W-IB2 using GA (Iris & Vestibular domains)

	IB2	**W-IB2**
	Iris	
Correct Classifications (%)	92.67	95.74
Standard deviation value	0.9944289818	0.0019296422
	Vestibular	
Correct Classifications (%)	77.50	89.98
Standard deviation value	1.7795131356	0.0037664172

Table 7. Best weight vector in the last generation

Number of Attribute	**Weight of Attribute**
1	0.9886000156
2	0.0610000006
3	0.9006999731
4	0.0653000026
5	0.0586999990
6	0.9714999794

Wine Domain

A similar situation to the one described previously occurred with IB2 in this domain. Table 8, third line, shows the performance values for IB2 and W-IB2 using GA, using the best weight vector in the 100th population, considering the thirteen attributes that describe this domain. Table 8, sixth line, shows the results of both algorithms, using the Reduced Wine domain. As can be seen both algorithms had their performances increased; the improvement of the W-IB2, however, was considerably higher compared to its performance in the complete domain.

Table 8. Performance of IB2 versus W-IB2 using GA (Wine & Reduced Wine domains)

	IB2	W-IB2
	Wine	
Correct Classifications (%)	71.66	66.29
Standard deviation value	2.1317701564	0.0003683525
	Reduced Wine	
Correct Classifications (%)	90.00	92.70
Standard deviation value	1.2292722549	0.0000035762

5. Conclusions

This paper describes how two lazy learning algorithms and a genetic algorithm can collaborate to produce better solutions than either lazy learning algorithm could produce by itself.

Based on the results obtained, we can say that the instance-based learning algorithms have a better performance when they are weighted by a weight vector found with the help of a GA. Although a genetic algorithm does not find optimal weight vectors, it has been shown that it is capable of finding vectors that are good enough to improve the performance of both instance-based algorithms.

In this line of research there are a number of issues that can be addressed in future work, including: to explore more possibilities of the genetic characteristics, such as other selection operators and the creep mutation operator; to focus on class-based attribute weight vectors and to try alternative values for k (in the k-NN). Different values for crossover and mutation rates should also be tried to determine to what extent they interfere in the results. A larger number of knowledge domains should also be considered in future experiments.

Acknowledgements.

To Prof. Dr. José F. Colafemina from the Service of Otoneurology of the Clinical Hospital of the Medical School of University of São Paulo in Ribeirão Preto for proving the Vestibular data and to Leonie C. Pearson whose comments helped us to greatly improve this article.

References

[1] Aha, D. W. Analysis of instance-based learning algorithms. *Proceedings of the 9th National Conference on Artificial Intelligence.* AAAI Press – The MIT Press. Vol. 02 (1991).

[2] Aha, D. W.; Kibler, D. & Albert, M. Instance-based learning algorithms. Machine Learning, 6, (1991), pp. 37-66.

[3] Beasley, D. et. al. An Overview of Genetic Algorithms: Part 1, Fundamentals, University Computing, v. 15, n. 4, (1993), pp. 170-181.

[4] Beasley, D. et. al. An Overview of Genetic Algorithms: Part 2, Fundamentals, University Computing, v. 15, n. 2, (1993), pp. 58-69.

[5] Blake, E. K.C.; Merz, C.J. UCI Repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html, (1998).

[6] Coley, D.A. An Introduction to Genetic Algorithms for Scientists and Engineers, World Scientific, (2001).

[7] Cover, T. and Hart, P. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory.* Vol. IT 13 (1967), pp. 21-27.

[8] Dasarathy, B.V. (ed.) Nearest neighbour (NN) norms: NN pattern classification techniques. Los Alamitos, CA: IEEE Computer Society Press (1991).

[9] Gates, G. W. "The reduced nearest neighbour rule". *IEEE Transactions on Information Theory*, vol. 18, pp. 431-433, (1972).

[10] Gen, M. and Cheng, R. Genetic Algorithms and Engineering Design. New York, John Wiley (1997).

[11] Goldberg, D. E. Genetic Algorithms in Search, Optimization and Machine Learning. Reading, U.S.A., Addison Wesley Publishing (1989).

[12] Holland, J. H. Adaptation in Natural and Artificial Systems. Ann Arbor, University of Michigan Press (1975).

[13] Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs. Berlin, Springer-Verlag (1992).

[14] Mitchell, T. M. Machine Learning, New York, McGraw-Hill (1997).

[15] Palma Neto, L.G., Figueira, L.B.; Nicoletti, M.C., Using a Family of Perceptron–Based Neural Networks for Detecting Central Vestibular System Problems. In: Proc. of The International Conference on Machine Learning and Applications, M. Wani (ed.), Los Angeles, CA, (2003), pp 193-199.

[16] Ramer, A. and Nicoletti, M. C. The symbolic side of a neuro-fuzzy system. *Studies in Fuzziness and Soft Computing.* P. Sincak and J. Vascak (eds), Physica-Verlag, Heidelberg. Vol. 54 (2000), pp. 447-452.

[17] Volpini, P., Figueira, L. B., Colafemina, J. F., Roque, A. C. A neural network-based system for the diagnosis of central vestibular lesion. In: Valafar, F. (Ed.). Proc. of the Int. Conf. on Mathematics and Engineering Techniques in Medicine and Biological Sciences-METMBS'02, CSREA Press, (2002), pp. 29-33.