Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks (AIE'96), pp. 11-14, 1996.

Instance-Based Learning with Genetically Derived Attribute Weights

D. Randall Wilson, Tony R. Martinez
e-mail: *randy@axon.cs.byu.edu*, *martinez@cs.byu.edu*
Computer Science Department, Brigham Young University, Provo, UT 84602, U.S.A.

Key words: instance-based learning, genetic algorithms, instance weights, generalization

Abstract. *This paper presents an inductive learning system called the Genetic Instance-Based Learning (GIBL) system. This system combines instance-based learning approaches with evolutionary computation in order to achieve high accuracy in the presence of irrelevant or redundant attributes. Evolutionary computation is used to find a set of attribute weights that yields a high estimate of classification accuracy. Results of experiments on 16 data sets are shown, and are compared with a non-weighted version of the instance-based learning system. The results indicate that the generalization accuracy of GIBL is somewhat higher than that of the non-weighted system on regular data, and is significantly higher on data with irrelevant or redundant attributes.*

1. Introduction

Much research has been directed at finding better ways of helping machines learn from examples. When domain knowledge in a particular area is weak, solutions can be expensive, time consuming and even impossible to derive using traditional programming techniques. Inductive machine learning techniques attempt to give machines the ability to learn from examples so that they can attain high accuracy at a low cost.

This paper addresses the problem of *classification*, in which an inductive learning system learns from a *training set, T*, which is a collection of examples, called *instances*. Each instance I in T has an *input vector* $\boldsymbol{x}$ and an *output class, c*. An input vector is made of *m input values*, labeled x_i ($1 \leq i \leq m$), one for each of m input variables (or *attributes*). The problem of classification in this context is to learn the mapping from an input vector to an output class in order to *generalize* to new examples it has not necessarily seen before.

Instance-based learning algorithms [1][2][3][4][5][6][7] are a class of supervised learning algorithms that retain some or all of the available training examples (or *instances*) during learning. During execution, a new input vector is compared to each stored instance. The class of the instance that is most similar to the new vector (using some distance function) is used as the predicted output class.

Instance-based learning algorithms are intuitive and simple, learn very quickly, and work well for many applications. However, they often have several drawbacks that cause them to generalize poorly on certain applications. In particular, their generalization accuracy usually degrades rapidly in the presence of irrelevant and redundant attributes. A system that can successfully deal with such attributes alleviates the need to carefully determine beforehand which attributes are needed in a set of data.

This paper seeks to address the problem of irrelevant and redundant attributes by using attribute weights to lessen the influence of such attributes. Section 2 discusses instance-based learning and the distance function used in the GIBL system, and provides motivation for finding attribute weights. Section 3 presents an evolutionary algorithm used to find attribute weights. Section 4 presents experimental results which indicate that attribute weights can significantly improve accuracy on data sets with irrelevant and redundant attributes and improve accuracy on regular data sets as well. Section 5 provides conclusions and several areas of future research.

2. Distance Function

Instance-based learning algorithms [1][2][3][4][5][6][7] need a distance function in order to determine which instance or instances are closest to a given input vector.

The original nearest neighbor algorithm typically uses the Euclidean distance function, which is defined as:

$$E(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2} \quad \text{(Eq. 1)}$$

Normalization. One weakness of the basic Euclidean distance function is that if one of the input variables has a relatively large range, then it can overpower the other input variables. For example, if an application has just two inputs, *x* and *y*, and *x* can have values from 1 to 1000, and *y* has values only from 1 to 10, then *y*'s influence on the distance function will usually be overpowered by *x*'s influence. Therefore, distances are often *normalized* by dividing the distance for each variable by the *range* (i.e., max-min) of that attribute, so that the distance for each input variable is in the range 0..1.

Another weakness of the Euclidean distance function is that it is not appropriate for some kinds of attributes. Therefore, the Genetic Instance-Based Learning system (hereafter "GIBL") uses a normalized heterogeneous distance function, as explained below.

Heterogeneous Distance Function. An attribute can be *linear*, such as a person's height or a temperature reading, or nominal. A *nominal* (or *symbolic*) attribute is one that can have only a discrete set of values, but whose values are not in any linear order. For example, a variable representing symptoms might have possible values of *headache, sore throat, chest pains, stomach pains, ear ache,* and *blurry vision.* Using a linear distance measurement on such values makes little sense in this case, so a function is needed that handles nominal inputs.

There are many applications that have both linear and nominal attributes and thus require a heterogeneous distance function. GIBL uses a function similar to that in [8]. The distance between two values *a* and *b* of a given attribute *i* is given as:

$$d_i(a,b) = \begin{cases} overlap(a,b), & \text{if } i \text{ is nominal} \\ difference_i(a,b), & \text{otherwise.} \end{cases} \quad \text{(Eq. 2)}$$

where *overlap* and *difference* are defined as:

$$overlap(a,b) = \begin{cases} 0, & \text{if } a = b \\ 1, & \text{otherwise} \end{cases} \quad \text{(Eq. 3)}$$

$$difference_i(a,b) = \frac{|a-b|}{range(i)} \quad \text{(Eq. 4)}$$

The function *range* is used to normalize the attributes, and is defined as:

$$range(i) = max(i) - min(i). \quad \text{(Eq. 5)}$$

where $max(i)$ and $min(i)$ are the maximum and minimum values, respectively, observed in the training set for attribute *i*. This means that it is possible for a new input vector to have a value outside this range and produce a difference value greater than one. However, the normalization serves to scale the attribute down to the point where differences are typically less than one.

The above definition for d_i returns a value which is (typically) in the range 0..1, whether the attribute is nominal or linear. The overall distance between two (possibly heterogeneous) input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is given as:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} w_i d_i(x_i, y_i)^2} \qquad \text{(Eq. 6)}$$

where m is the number of attributes, d_i is the distance function given in Equation 2 and w_i is an attribute weight used to weight the distance along each dimension.

The distance d_i for each attribute i is squared in order to make the distance function behave like Euclidean distance in the presence of continuous attributes. Note that the square root in Equation 6 is not performed in practice, because it does not alter the ordering of closeness among neighbors.

Note that when the attribute weights are all equal (e.g., $w_{1..m}$=1.0), this distance function becomes equivalent to normalized Euclidean distance for linear attributes. Normalization causes the weight of each attribute to become equal and removes the arbitrary weighting due to scale. However, some attributes may be more useful in others in determining the output class, and other attributes may even be damaging to classification accuracy. Appropriate attribute weights can alter the decision boundaries in the input space in order to reduce the damaging effect of irrelevant and redundant attributes, while fine-tuning weights on useful attributes as well.

Irrelevant Attributes. When applying a learning system to a real application, it is not always obvious which input attributes would be useful in finding a solution. It would often be advantageous to provide as much information as possible on the problem and then allow the learning system to automatically determine which attributes are relevant.

One problem with unweighted instance-based learning systems is that they are very susceptible to irrelevant attributes. For example, consider an application in which several measurements, including blood pressure, are taken on a patient, and the system is to determine whether the patient has a particular disease. It may turn out that blood pressure has little or nothing to do with the disease.

In such a case the irrelevant attribute adds a somewhat random value to the measurement of distance between two instances or input vectors. This can make it so that an instance or node which *should* appear close to an input vector (and thus classify it correctly) may appear farther away, resulting in a more random classification and lower classification accuracy.

Classification accuracy can often be restored by assigning irrelevant attributes small weights to reduce their effect on the distance function. Reducing the weight of an attribute all the way to zero would effectively remove it altogether.

Redundant Attributes. The effect of redundant attributes is more subtle than that of irrelevant attributes, and has much in common with the scaling problem that necessitates normalization. An attribute is redundant if it can be derived from the values of the remaining attributes. The simplest example of a redundant attribute is one that is repeated one or more times. For example, if an attribute were repeated 10 times in a data set, the distances summed over these attributes would get 10 times the weight of the distance along each of the remaining dimensions. This would result in the same classification as using the repeated attribute once with a weight that is 10 times that of the other attributes.

In real applications, attributes are not often repeated explicitly, but it is not uncommon for some of the attributes to add no new information that cannot be derived from the others. For example, one attribute can be the squared value of another attribute, or the average of several other attributes. This causes too much credit to be assigned to one aspect of the problem, even if the weights of the individual attributes are equal.

It should be noted that it is not necessarily bad for one attribute to have a higher weight than another. If one attribute is in fact more relevant or useful than another then it may be quite useful to assign it a higher weight. However, giving an attribute a higher weight just because it happened to be recorded twice in slightly different forms (or some other form of redundancy) is an arbitrary policy. Just as normalization overcomes the effect of arbitrary weighting due to differing ranges, it would typically be better to correct for redundancy, and assign weights based on less arbitrary criteria.

Redundant attributes often have a correlation with the output class and thus cannot always be detected by some techniques that can identify irrelevant attributes [9]. However, one way to deal with irrelevant attributes and redundant attributes is to find weight settings for each attribute that seems to improve classification. Wettschereck, Aha and Takao [9] provide an excellent review of many attribute weighting schemes (as well as other kinds of weighting schemes). One of their conclusions is that systems which use performance feedback to decide on the values of weights tend to achieve higher accuracy than those that do not. Section 3 presents an evolutionary algorithm which uses performance feedback (in the form of classification accuracy estimation) in order to find attribute weights for an instance-based learning system.

3. Evolutionary Algorithm

This section presents a method of discovering and evaluating the attribute weights w_i of Equation 6 through evolutionary algorithms and instance-based techniques.

This problem can be viewed as an optimization problem. Specifically, the problem is to find a vector $\boldsymbol{w}$ of real-valued weights $w_1..w_m$, such that classification accuracy is as high as possible (where m is again the number of attributes in a given application or data set). Unfortunately, the weight space is infinite (or nearly so, within precision limits). Even if only a few (e.g., 10) different values are allowed for consideration for each weight, the size of the weight space increases exponentially with the number of attributes (e.g., 10^m if 10 values are used).

Evolutionary Algorithms [11] provide heuristics which can aid in searching an intractable space. A *population* of *individuals* is initialized to random places in the search space, and each individual is evaluated and given a *fitness* score. Those individuals with the highest fitness score are chosen to be parents of the next generation of individuals. Genetic operators such as recombination and mutation are used to modify or combine parents into new individuals that are similar enough to their parents that they have a good chance of being as good as their parents, but different enough that they also have a chance of being better.

The Genetic Instance-Based Learning (GIBL) Algorithm uses genetic operators to guide the search through the weight space, and instance-based techniques to evaluate each weight setting and determine its fitness. Figure 1 gives a pseudo-code algorithm for the weight learning scheme used in GIBL.

GIBL uses a population size of 40, and initializes its population randomly with the exception of one individual that has all of its weights set to 1.0. This allows one of the starting points of the search through the weight space to be the "default" equally-weighted setting. The GIBL system uses a vector of real values as its representation, and genetic operators work directly with these values.

The system allows individuals to survive for only one generation, and replaces the entire generation with a new one after each evaluation of the population. However, the best weight setting found by any individual in any generation (i.e., the one with the highest fitness) is saved separately, and updated whenever another individual has an even higher fitness. In this way the best solution is not lost because of the lack of survival, and the entire population can be utilized for finding new weight vectors to explore. GIBL continues until it has not improved upon its best weight setting for ten generations.

```
InitializePopulation(P);
while (time_since_improvement < termination_criteria)
{ increment time_since_improvement;

  /* Evaluate population */
  for i=1 to population_size
  { EvaluateIndividual(P[i]);
    if P[i] is the best individual seen so far
      { save a copy of P[i]'s weights
        time_since_improvement=0;
      } /* if */
  } /* for */

  /* Create New Population C */
  for child = 1 to population_size
  { /* Create a new child in C */
    parent = PickParent(P);
    if (rnd()> recombination_rate)
      C[child] = Recombine(parent,PickParent(P))
    else C[child] = Mutate(parent);
  } /* for */

  /* Make child population the new parent population. */
  P = C;
} /* while */
```

Figure 1. GIBL weight-learning algorithm.

Genetic Operators. *Recombination* and *mutation* are both used in GIBL as genetic operators. A *recombination rate* (set to .3 in GIBL) is used to decide what proportion of selected parents are combined with another parent by selecting each weight randomly from one of the two parents and the remaining parents are mutated instead. For those parents being mutated, a *mutation rate* (set at .5) is used to determine what the chance is of each weight being mutated, and a *step size* (set at .2) is used as the standard deviation of a normally-distributed value (with a mean of 0) which is added to the current weight.

Parent Selection. GIBL selects parents probabilistically, based on their fitness scores. That is, those with higher fitness scores are more likely to be used in creating a new individual than those with lower fitness scores. This allows each generation to be created mostly from good individuals, thus guiding the search in positive directions. However, it also allows each individual to have some chance at being selected, thus allowing some diversity to remain in the population. This helps prevent the population from becoming a collection of nearly identical individuals.

One drawback with using probabilistic parent selection is that if the individuals in the population all start to have similar fitness scores, then the search is not strongly directed towards more fit individuals, and will therefore take more random (and less productive) search paths.

In order to make good individuals much more likely to be chosen than those that are less fit, the fitness scores are passed through a spreading function which raises the values to the fourth power. This function was found empirically to produce reasonable results. Figure 2 shows how this affects the probability of 10 individuals being chosen.

ID	Original fitness	Probability of selection	New fitness	New probability of selection
1	0.26	5.7 %	0.00483	0.6 %
2	0.30	6.5 %	0.00822	1.0 %
3	0.32	6.8 %	0.01026	1.3 %
4	0.38	8.2 %	0.02141	2.7 %
5	0.43	9.2 %	0.03428	4.3 %
6	0.44	9.5 %	0.03874	4.9 %
7	0.52	11.2 %	0.07482	9.5 %
8	0.63	13.5 %	0.15623	19.8 %
9	0.64	13.8 %	0.17077	21.6 %
10	0.72	15.5 %	0.27077	34.3 %

Figure 2. Spreading function modifying parent selection.

The individuals in Figure 2 are sorted by fitness score. On the left is shown the original fitness scores along with the probability of each individual being chosen as a parent during each parent selection. On the right is shown the new fitness score, which is the old score raised to the fourth power. Figure 3 illustrates these same percentages graphically.

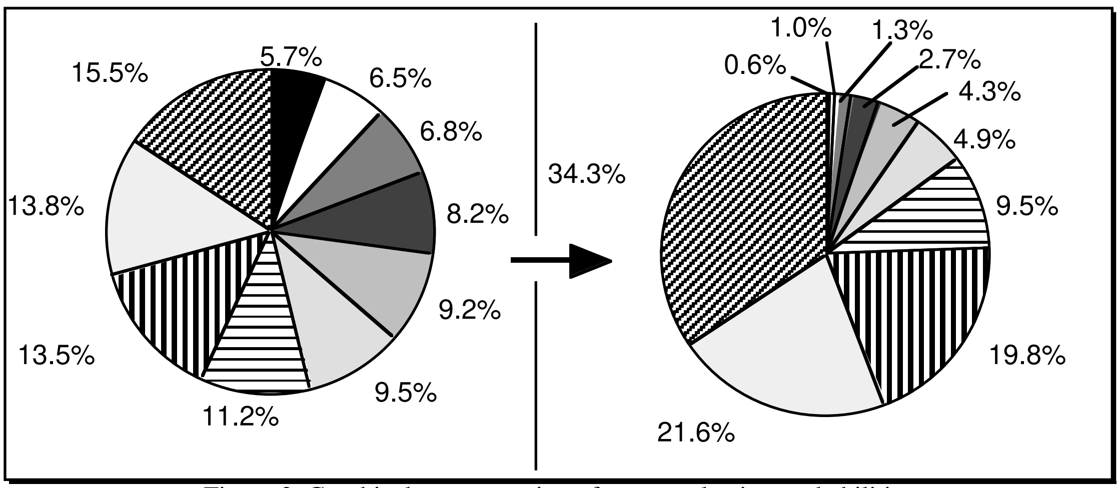


Figure 3. Graphical representation of parent selection probabilities.

Before applying the spreading function, the probabilities of selection for each individual are fairly evenly spaced. As can readily be seen, the spreading function makes the best few individuals much more likely to be chosen than the remaining individuals, though it does give each individual at least a small chance of being selected.

Evaluation Function. The evaluation function is crucial to the operation of an evolutionary algorithm. It assigns a fitness value to each individual, which in turn is used to decide which individuals to use to create individuals in subsequent generations. The fitness value does not necessarily need to be precise, because it is used to probabilistically choose parents anyway, but it must be at least approximately correct most of the time in order for the search to proceed effectively.

In the GIBL algorithm, the fitness represents an estimate of how accurate classification will be on an application using a given weight vector. In order to test the final, "best" weight settings, part of the available data, called the *test set*, must be held out and not used by any portion of the learning algorithm, so it is not appropriate to use the test set to evaluate a weight vector.

Therefore, weight vectors are tested using leave-one-out cross-validation on the training set itself. That is, for each instance *I* in the training set *T*, the nearest instance *other than the instance itself*, i.e., $N \in (T - I)$, is found, using the weight vector in question, to see if *N* is of the same class as *I*. The fitness is then the ratio (raised to the fourth power, as explained above) of correct classifications to total classification attempts. Pseudo-code for the evaluation function is given in Figure 4 below.

```
EvaluateIndividual(individual P[i])
{ /* Given an individual P[i], find its fitness value.*/
  for each instance t in training set T
    { nearest_neighbor = FindNearest(t,P[i].weights);
      if (SameClass(nearest_neighbor,t))
        then increment correct;
      increment total;
    } /* for */
  P[i].fitness = (correct / total)^4;
}
```

Figure 4. Evaluation function algorithm.

Unfortunately, the straightforward implementation of this evaluation function is an $O(n^2)$ operation, where n is the number of instances in the training set. Since the evaluation function is used on every individual in every generation, this can quickly become a slow process as the number of instances grows.

One thing that can be done to speed up this evaluation is to simply limit the number of training set instances used in the evaluation function, i.e., the number of times the nearest neighbor of an instance is found. If the number of instances available becomes quite large, it may not be necessary to use all of them before a reasonably confident fitness score can be derived. Current research is seeking to determine just how many instances are required to provide acceptable fitness estimates. The results presented in this paper make use of all available instances, but the GIBL system allows the user to specify what proportion of the available instances to use in the evaluation function.

The GIBL system also uses a technique called *projection* [12] to reduce the number of distance calculations that must be performed before the nearest neighbor can be found.

4. Experiments

The GIBL algorithm was designed to handle irrelevant and redundant attributes, but it is important to make sure that it does not trade success in these areas for poor performance on regular data. This section presents empirical results that indicate that GIBL performs slightly better than a non-weighted version on regular data, and significantly better on data with irrelevant or redundant attributes.

The GIBL algorithm was implemented and tested on 16 databases from the Machine Learning Database Repository at the University of California Irvine [13]. Each test consisted of ten trials, each using one of ten partitions of the data randomly selected from the data sets, i.e., 10-fold cross-validation. Each trial consisted of building a training set using 90% of the available data, initializing the population, and evaluating populations until 10 generations passed without any improvement.

The best weight setting found during the trial was then used in classifying each instance in the test set (i.e., the remaining 10% of the data). The classification accuracy on the test set for each trial was recorded and compared to the default accuracy, i.e., that obtained by using no weights. The non-weighted algorithm uses the same test sets and training sets, and the same distance function, except that all weights are set to 1.0.

The average accuracy for each database over all trials is shown in Figure 6. One asterisk (*) indicates that the higher value is statistically significantly higher at a 90% confidence level, using a one-tailed paired *t*-test. Two asterisks (**) are used to mark differences that are significant at a 95% confidence interval.

Note that these data sets do not necessarily have irrelevant or redundant attributes, but are provided to see how the algorithm performs on regular data. GIBL had a significantly higher accuracy in five out of the sixteen data sets, and was significantly lower in only one case. This indicates that GIBL does somewhat better than the non-weighted algorithm when there are no particularly irrelevant or redundant attributes present.

Database	GIBL	Non-weighted
Australian Credit	80.72	81.88
Bridges	59.18**	52.73
Credit Screening	81.45	81.16
Echocardiogram	52.25	53.02
Flag	55.11**	48.29
Glass	78.57**	70.52
Heart (Cleveland)	49.60	48.59
Heart (Hungarian)	73.80	77.20**
Heart (Swiss)	20.38	28.40
Image Segmentation	93.33	93.57
Iris	94.67	95.33
Led-Creator	64.80**	51.80
Liver Disorders	67.25	63.47
Pima Indians Diabetes	70.05	70.31
Vowel	98.10	98.86
Wine	96.60*	95.46
Average:	71.02	69.64

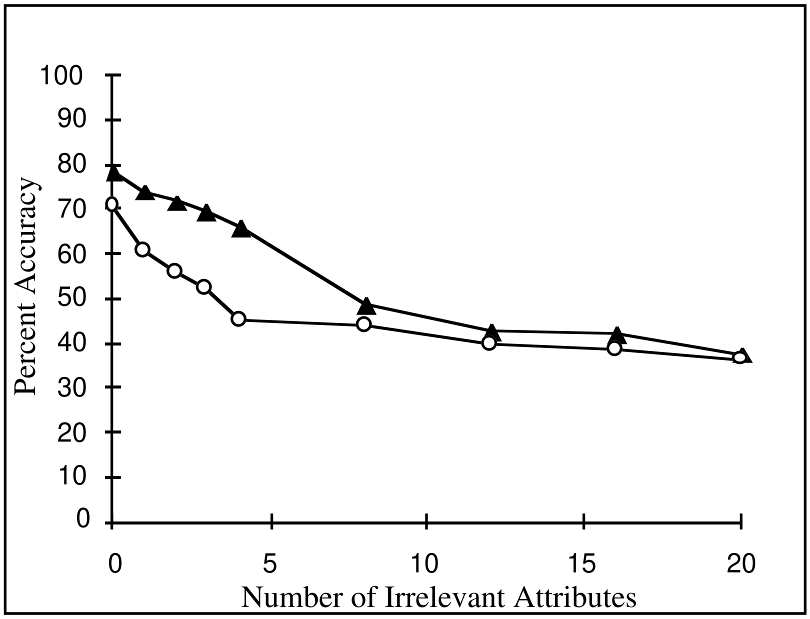
Figure 6. Experimental results on data sets without irrelevant or redundant attributes.

Testing Irrelevant Attributes. In order to determine whether GIBL improves performance in the face of irrelevant attributes, a real data set was artificially modified to see what effect this would have on the accuracy of the default rule compared to GIBL.

Irrelevant attributes were added to the Glass database in order to see the effect on classification accuracy. The original glass database has nine continuous input attributes. As irrelevant (completely random) attributes are added to the database, the classification accuracy is expected to fall for nearly any algorithm. However, the rate at which the accuracy falls distinguishes algorithms which are robust in the presence of irrelevant attributes from those which are sensitive to them. Figure 7 summarizes the results of these experiments.

As the number of irrelevant attributes is increased, the non-weighted algorithm quickly degrades below the 50% accuracy level, while the GIBL system remains much more accurate. After the addition of twenty irrelevant attributes, GIBL, too, dips in accuracy to nearly the level of the unweighted algorithm.

As shown in Figure 7, the GIBL's performance never drops as low as the default rule, and is significantly higher than the non-weighted algorithm for most of the settings.



Glass Database

Irrelevant Attributes	GIBL	Non-weighted
0	78.57**	70.52
1	74.29**	60.82
2	72.01**	56.13
3	69.68**	52.36
4	65.91**	45.24
8	48.68	43.96
12	42.99	39.87
16	42.10	38.83
20	37.36	36.41

Figure 7. Glass data with irrelevant attributes.

Testing Redundant Attributes. To test the effect of redundant attributes, the glass database was again modified, this time by repeating one of the attributes several times. The results of this series of experiments is summarized in Figure 8.

GIBL was able to remain significantly more accurate than the default rule as redundant attributes were added to the data. The added redundant attributes contained valid data, as opposed to pure noise as in the case of irrelevant attributes. Therefore, both GIBL and the non-weighted model were able to maintain higher accuracy than in the presence of irrelevant attributes, even with the addition of many (19) copies of the same attribute.

5. Conclusions & Future Research

The Genetic Instance Based Learning (GIBL) System presented in this paper was designed to be robust in the presence of irrelevant and redundant attributes, and to fine-tune weights in order to improve classification accuracy even on data sets without such attributes. In the experiments presented in this paper, its classification performance on regular data sets is somewhat higher that of the non-weighted algorithm in the above experiments. Furthermore, on data sets with irrelevant and redundant attributes, its accuracy remains significantly higher than the non-weighted algorithm.

The improved accuracy is attained at the cost of increased processing time during the learning phase of the algorithm. However, once the attribute weights are derived, execution time proceeds at the same speed as the nonweighted instance-based system. Current research includes the following:

Glass Database

Redundant Attributes	GIBL	Non-weighted
0	78.57**	70.52
1	75.22**	70.04
2	76.56**	68.66
3	76.15**	68.66
7	73.33**	67.79
11	74.76**	65.00
15	69.63**	64.50
19	71.49**	64.03
99	64.03*	60.80

Figure 8. Glass data with redundant attributes.

• Determining how many instances need to be examined in the evaluation function before the fitness value is reliable enough, in order to reduce training time;

- Finding out how many individuals are sufficient in the population;
- Discovering good settings for the various parameters in the system (namely, the recombination rate, mutation rate, mutation step size);
- Combining new heterogeneous distance functions with the genetically-derived weights; and
- Examining methods of reducing the number of instances that need to be stored while maintaining reasonable classification accuracy.

The results of this research are encouraging. They show that attribute weights can be used to improve generalization accuracy and successfully resist the damaging effects of irrelevant and redundant attributes in instance-based learning systems.

Bibliography

[1] Cover, T. M., and P. E. Hart, (1967). "Nearest Neighbor Pattern Classification," *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, vol. 13, no. 1, January 1967, pp. 21-27.

[2] Dasarathy, Belur V., (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, Los Alamitos, CA: IEEE Computer Society Press.

[3] Aha, David W., Dennis Kibler, Marc K. Albert, (1991). "Instance-Based Learning Algorithms," *Machine Learning*, vol. 6, pp. 37-66.

[4] Aha, David W., (1992). "Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms," *International Journal of Man-Machine Studies*, vol. 36, pp. 267-287.

[5] Cost, Scott, and Steven Salzberg, (1993). "A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features," *Machine Learning*, vol. 10, pp. 57-78.

[6] Domingos, Pedro, (1995). "Rule Induction and Instance-Based Learning: A Unified Approach," to appear in *The 1995 International Joint Conference on Artificial Intelligence (IJCAI-95).*

[7] Stanfill, C., and D. Waltz, (1986). "Toward memory-based reasoning," *Communications of the ACM*, vol. 29, December 1986, pp. 1213-1228.

[8] Giraud-Carrier, Christophe, and Tony Martinez, (1995). "An Efficient Metric for Heterogeneous Inductive Learning Applications in the Attribute-Value Language," *Intelligent Systems*, pp. 341-350.

[9] Wilson, D. Randall, (1994). *Prototype Styles of Generalization*, Master's Thesis, Brigham Young University.

[10] Wettschereck, Dietrich, David W. Aha, and Takao Mohri, (1995). "A Review and Comparative Evaluation of Feature Weighting Methods for Lazy Learning Algorithms," Technical Report AIC-95-012, Washington, D.C.: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence.

[11] Spears, William M., Kenneth A. De Jong, Thomas Bäck, David B. Fogel, and Hugo de Garis, (1993). "An Overview of Evolutionary Computation," *Proceedings of the European Conference on Machine Learning*, vol. 667, pp. 442-459.

[12] Papadimitriou, Christos H., and Jon Louis Bentley, (1980). "A Worst-Case Analysis of Nearest Neighbor Searching by Projection," *Lecture Notes in Computer Science*, vol. 85, Automata Languages and Programming, pp. 470-482.

[13] Murphy, P. M., and D. W. Aha, (1993). *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science.