Proceedings of the International Symposium on Neuroinformatics and Neurocomputers, pp. 238-45, 1995

Using Multiple Statistical Prototypes to Classify Continuously Valued Data

Dan Ventura and Tony R. Martinez

Computer Science Department, Brigham Young University, Provo, Utah 84602
e-mail: dan@axon.cs.byu.edu, martinez@cs.byu.edu

Multiple Statistical Prototypes (MSP) is a modification of a standard minimum distance classification scheme that generates multiple prototypes per class using a modified greedy heuristic. Empirical comparison of MSP with other well-known learning algorithms shows MSP to be a robust algorithm that uses a very simple premise to produce good generalization and achieve parsimonious hypothesis representation.

1. Introduction

The idea of using prototypes to represent classes has proven to be a powerful mechanism for learning [1][2][14][12][9]. It is a simple and natural approach to the problem of dealing with continuously valued attributes. The basic assumption is that given an m-dimensional space defined by the input variables, there exists one or more representative points in that space for each output class. These representative points are termed prototypes. The multiple statistical prototypes algorithm (MSP) is a simple variation on this idea. It assumes that all input variables are continuously valued and that each output class can be represented by one or more gaussian functions over these input variables. This assumption is not unreasonable because all functions may be approximated by one or more gaussian bases, the worst case being the degenerate one.

The idea of using statistical information obtained from a training set in the formation of prototypes has also been used in other models. Two examples of similar systems are radial basis function networks [8] and CLASSIT [6]. MSP differs from CLASSIT in its supervised approach to learning, its utility measure (distance metric), and in the fact that MSP does not use a merge-type operation. MSP and RBF both use prototypes to perform a non-linear mapping from the input space to the output space. However, they differ in their manner of calculating prototypes and in their mapping function.

Section two presents the basic statistical prototypes (SP) which employ a single prototype per class. Section three extends this to multiple prototypes per class (MSP). Both sections include empirical results and comparisons with other algorithms. Section four provides further empirical results and analysis and section five concludes the paper.

2. Creating Statistical Prototypes

Initially, each output class is assumed to be represented by a single m-dimensional gaussian base over the input space. Therefore, by assumption, each output class is represented with a single prototype. Define:

T as a set of training instances;
n as the number of instances in T, that is $n = |T|$;
c as the number of output classes represented in T;
v as the number of input variables represented in T;
i as an index that ranges $0 \le i < c$ and indicates output class;
j as an index that ranges $0 \le j < v$ and indicates input variable;
T_i as the *ith* sub-training set obtained by partitioning T by output class;
o_i as the *ith* output class of T;
m_{ij} as the mean of the *jth* input variable for T_i;
σ_{ij}, as the standard deviation of the *jth* input variable for T_i;
$\boldsymbol{p}_i$ as the prototype for class i;
$\boldsymbol{x}$ as a vector of inputs representing an instance;
x_j as the *jth* input of an instance;
d_i as the normal distance between a point $\boldsymbol{x}$ and $\boldsymbol{p}_i$.

The basic algorithm (algorithm SP in figure 1) is outlined below and proceeds as follows. First, divide the training set T by output class, creating c sub-training sets T_i. For each T_i and each input variable j calculate m_{ij} and σ_{ij}. $\boldsymbol{p}_i$ is the vector of ordered pairs (m_{ij}, σ_{ij}). Classification of a new instance $\boldsymbol{x}$ is accomplished by calculating all d_i (the distance from each prototype) for the point defined by that instance and outputting z such that $d_z = min\{d_i\}$, where d_i is calculated as

$$d_i = \sum_{j=0}^{v} \frac{|x_j - m_{ij}|}{\sigma_{ij}}.$$

```
Learn()
    Create subtraining sets $T_i$
    Calculate $m_{ij}$ and $\sigma_{ij}$
    Create $\boldsymbol{p}_i$ as $j$ ordered pairs, ($m_{ij}$, $\sigma_{ij}$)
Classify($\boldsymbol{x}$)
    calculate all $d_i$
    output $z$ such that $d_z = min\{d_i\}$
```

Figure 1. Single prototype per output class (algorithm SP)

Two points are worth a brief mention here. First, this is obviously a variation of a standard minimum distance classifier. Thus, SP is not a novel idea; however, it is important as background material because it is the basis for MSP, which extends the minimum distance classifier to multiple prototypes per class in a novel way. Second, the name statistical prototypes may be somewhat misleading. SP is not a statistical approach to clustering/classification in the Bayesian sense of relying on the statistical properties of the underlying distributions. It is statistical only in the sense of employing the simple statistical variables of mean and standard deviation in representing prototypes.

2.1. An Example. Suppose that T consists of the following set of nine instances:

4.8	1.8	->	negative,	3.5	1.2	->	positive,
5.0	1.9	->	negative,	3.0	1.1	->	positive,
4.5	1.7	->	negative,	3.9	1.4	->	positive,
3.5	1.2	->	negative,	4.2	1.5	->	positive.
3.6	1.2	->	negative,				

Then c = 2, v = 2, o_0 is negative, and o_1 is positive. T_0 consists of the first five instances, and T_1 of the last four instances. Calculating the means and standard deviations we get

$$m_{00} = 4.28, \quad m_{01} = 1.56, \quad m_{10} = 3.65, \quad m_{11} = 1.30,$$
$$\sigma_{00} = .618, \quad \sigma_{01} = .301, \quad \sigma_{10} = .450, \quad \sigma_{11} = .158.$$

The notation used throughout for expressing a prototype is

$$\boldsymbol{p}_i = \begin{matrix} m_{i0} & m_{i1} & \dots & m_{iv-1} & \\ \sigma_{i0} & \sigma_{i1} & \dots & \sigma_{iv-1} & \rightarrow o_i. \end{matrix}$$

Therefore,

$$\boldsymbol{p}_0 = \begin{matrix} 4.280 & 1.560 & \\ 0.618 & 0.301 & \rightarrow \text{ negative,} \end{matrix}$$

$$\boldsymbol{p}_1 = \begin{matrix} 3.650 & 1.300 & \\ 0.450 & 0.158 & \rightarrow \text{ positive.} \end{matrix}$$

Now, suppose an instance $\boldsymbol{x}$ = (5.0, 1.1) is presented to the system for classification. Distances from $\boldsymbol{x}$ to each prototype are calculated:

$$d_0 = \frac{|5 - 4.28|}{.618} + \frac{|1.1 - 1.56|}{.301} = 2.69,$$

$$d_1 = \frac{|5 - 3.65|}{.618} + \frac{|1.1 - 1.3|}{.158} = 4.27.$$

The prototype for o_0 (negative) is the closest to the example so $\boldsymbol{x}$ is classified as negative.

2.2. Empirical Results. SP was compared empirically with the following nine well-known learning models: ID3 [11], C4.5 (tree and rule versions) [10], Cart [3], Bayes [3], MML [13], IB3 [1], IB4 [2], and CN2 [4]. The results are shown in Table 1. These results represent averages over 25 runs. The first column shows the percentage of test instances classified correctly using SP. As before, 70% of the data were used as training instances, and the remaining 30% as test instances. The remaining three columns give the best, average and worst performance of the nine algorithms mentioned above and were obtained from [18].

Dataset	SP	Best	Average	Worst
Wine	**94.6**	93.8	92.6	90.5
Vowel	**46.8**	92.6	81.8	73.3
Sonar	**73.3**	78.9	75.7	72.5
Iris	**95.0**	95.3	94.7	94.0
Bupa	**59.9**	67.0	62.8	56.5
Pima	**69.1**	74.1	71.1	68.3
Vehicle	**47.5**	77.6	70.1	59.2
Thyroid	**94.5**	93.5	92.1	89.2
Glass	**49.3**	68.6	66.4	61.6
Waveform	**79.5**	72.0	68.7	66.0
Wavenoise	**81.6**	73.0	68.8	65.0

Table 1. Using single statistical prototypes to classify the data sets

Notice that SP does very well in comparison with other learning models on the iris, thyroid, waveform, wavenoise, and wine data sets. Since a single prototype represents each class this indicates that these are fairly simple data sets to learn and represent essentially linearly separable problems. SP performs fairly well on the bupa, pima, and sonar data sets. This indicates that these problems, too, are fairly easy although not quite linearly separable, since other, more advanced, methods perform significantly better on them. Creating multiple prototypes per class could improve performance. Finally, SP does quite poorly on the glass, segment, vehicle, and vowel data sets. It appears that these data sets are far from linearly separable, and thus a single prototype per class representation would be expected to perform poorly. Multiple prototypes per class are a necessity in these cases.

3. Extension to Multiple Prototypes Per Class (MSP)

Extending single-prototype minimum distance classifiers to multiple prototypes is a difficult problem because there exists no optimal method for determining which prototypes to use. Various approaches to the problem include neural networks such as competitive learning [14], counter propagation networks [9], and RCE [12] as well as instance based learning [1][2], nearest neighbor methods [5], and clustering techniques such as ISODATA[16]. MSP employs an error feedback heuristic to extend the simple statistical prototype described in the previous section. Add the new definitions:

ε_b as the error in classifying the training set before creating a new prototype;

ε_a as the error in classifying the training set after temporarily splitting a prototype;

ε_{small} as the smallest error in classifying the training set after temporarily splitting each prototype;

q_i as the number of prototypes for output class i;

k as an index that ranges $0 \le k < q_i$ and indicates prototype number;

i_{small}, j_{small}, k_{small} as the values of i, j, and k during temporary splitting of prototypes that result in the value ε_{small};

q_{small} as $q_{i_{small}}$, a slight abuse of notation;

T_i^k as the sub-training set that contains all the instances of output class i that are closest to the kth prototype for class i;

t_i^k as an instance in T_i^k;

$min\{j, T_i^k\}$ as the minimum value of input j for any t_i^k in T_i^k;

$max\{j, T_i^k\}$ as the maximum value of input j for any t_i^k in T_i^k.

Also, replace the definitions for $\boldsymbol{p}_i$ and d_i by the following:

$\boldsymbol{p}_i^k$ as the kth prototype for class i;

d_i^k as the normal distance between a point and $\boldsymbol{p}_i^k$.

A method for determining how many prototypes are required for an output class is required. MSP applies a two level greedy algorithm as follows (figure 2).

First, run SP to generate a single prototype, $\boldsymbol{p}_i^0$, for each output class. Next, calculate ε_b by using the prototypes to attempt to classify the training set T. Then temporarily split $\boldsymbol{p}_0^0$ on input 0 as follows. Find $min\{0, T_0^0\}$ and $max\{0, T_0^0\}$ and temporarily divide T_0^0 into T_0^{0min} and T_0^{0max} where T_0^{0min} contains all t_0^0 closer to $min\{0, T_0^0\}$ than to $max\{0, T_0^0\}$, and T_0^{0max} contains all t_0^0 closer to $max\{0, T_0^0\}$ than to $min\{0, T_0^0\}$. Calculate temporary prototypes $\boldsymbol{p}_0^{0min}$ and $\boldsymbol{p}_0^{0max}$ as before and then calculate ε_a by again attempting to classify T, the only change being that $\boldsymbol{p}_0^0$ has been temporarily replaced by $\boldsymbol{p}_0^{0min}$ and $\boldsymbol{p}_0^{0max}$. Reunite $\boldsymbol{p}_0^{0min}$ and $\boldsymbol{p}_0^{0max}$ into $\boldsymbol{p}_0^0$ and repeat for all inputs x_j and all prototypes $\boldsymbol{p}_i^k$, in order to find the values i_{small}, j_{small}, k_{small}, and ε_{small}. Now, if $\varepsilon_{small} < \varepsilon_b$, permanently split $\boldsymbol{p}_{i_{small}}^{k_{small}}$ into $\boldsymbol{p}_{i_{small}}^{k_{small}}$ and $\boldsymbol{p}_{i_{small}}^{q_{small}}$, increment q_{small}, and set $\varepsilon_b = \varepsilon_{small}$. If $\varepsilon_{small} \ge \varepsilon_b$, *temporarily* split $\boldsymbol{p}_{i_{small}}^{k_{small}}$ into $\boldsymbol{p}_{i_{small}}^{k_{small}}$ and $\boldsymbol{p}_{i_{small}}^{q_{small}}$, *temporarily* increment q_{small}, *temporarily* set $\varepsilon_b = \varepsilon_{small}$, and repeat the entire process. If again $\varepsilon_{small} \ge \varepsilon_b$, unsplit $\boldsymbol{p}_{i_{small}}^{k_{small}}$, decrement q_{small}, and quit. Otherwise, make the temporary split permanent, permanently split the new $\boldsymbol{p}_{i_{small}}^{k_{small}}$ into $\boldsymbol{p}_{i_{small}}^{k_{small}}$ and $\boldsymbol{p}_{i_{small}}^{q_{small}}$, increment q_{small}, set $\varepsilon_b = \varepsilon_{small}$, and repeat until $\varepsilon_{small} \ge \varepsilon_b$ for two consecutive passes through the training set. Classification of new instances is performed the same as in SP.

```
Learn()
    Run SP
    Calculate $\varepsilon_b$
    While(not $\varepsilon_{small} \geq \varepsilon_b$ for two consecutive passes)
        For $i$ = 0 to $c$-1
            For $k$ = 0 to $q_i$-1
                For $j$ = 0 to $v$-1
                    Split( $\boldsymbol{p}_i^k$ ,$j$)
                    Calculate $\varepsilon_a$
                    If $\varepsilon_a < \varepsilon_{small}$
                        $\varepsilon_{small} = \varepsilon_a$, $i_{small} = i$, $j_{small} = j$, $k_{small} = k$
                    Unsplit( $\boldsymbol{p}_i^k$ ,$j$)
        If $\varepsilon_{small} < \varepsilon_b$
            Split( $\boldsymbol{p}_{i_{small}}^{k_{small}}$ ,$j_{small}$)
            $q_{small} = q_{small}+1$
            $\varepsilon_b = \varepsilon_{small}$

Split( $\boldsymbol{p}_i^k$ ,$j$)
    $T_i^{kmin} = \{ t_i^k \mid t_i^k$ is closer to $min\{j, T_i^k\}$ than to $max\{j, T_i^k\}\}$
    $T_i^{kmax} = \{ t_i^k \mid t_i^k$ is closer to $max\{j, T_i^k\}$ than to $min\{j, T_i^k\}\}$
    calculate $\boldsymbol{p}_i^{kmin}$ and $\boldsymbol{p}_i^{kmax}$
    replace $\boldsymbol{p}_i^k$ with $\boldsymbol{p}_i^{kmin}$ and $\boldsymbol{p}_i^{kmax}$

Unsplit( $\boldsymbol{p}_i^{kmin}$, $\boldsymbol{p}_i^{kmax}$)
    $T_i^k = T_i^{kmin} \cup T_i^{kmax}$
    calculate $\boldsymbol{p}_i^k$
    replace $\boldsymbol{p}_i^{kmin}$ and $\boldsymbol{p}_i^{kmax}$ with $\boldsymbol{p}_i^k$

Classify($\boldsymbol{x}$)
    calculate all $d_i$
    output $z$ such that $d_z = min\{d_i\}$
```

Figure 2. Multiple prototypes per output class (algorithm MSP)

3.1. An Example. Continuing with the previous example, recall that the initial calculation of prototypes yielded

$$\boldsymbol{p}_0^0 = \begin{array}{cc} 4.280 & 1.560 \\ 0.618 & 0.301 \end{array} \quad -> \quad \text{negative,}$$

$$\boldsymbol{p}_1^0 = \begin{array}{cc} 3.650 & 1.300 \\ 0.450 & 0.158 \end{array} \quad -> \quad \text{positive.}$$

The distances from each prototype for each instance of the training set are presented in the following table:

		d_0	d_1
(4.8,	1.8)	1.639	5.720
(5.0,	1.9)	2.295	6.797
(4.5,	1.7)	0.821	4.421
(3.5,	1.2)	2.459	0.966
(3.6,	1.2)	2.296	0.744
(3.5,	1.2)	2.459	0.966
(3.0,	1.1)	3.599	2.710
(3.9,	1.4)	1.146	1.411
(4.2,	1.5)	0.329	2.488

Now, T is classified using these prototypes and the results are shown below with the correct classification in parentheses.

4.8	1.8	->	negative,	3.5	1.2	->	positive ,
5.0	1.9	->	negative,	3.0	1.1	->	positive ,
4.5	1.7	->	negative,	3.9	1.4	->	negative*,
3.5	1.2	->	positive *,	4.2	1.5	->	negative*.
3.6	1.2	->	positive *,				

* misclassified

Thus, with a single prototype per class, ε_b = 4/9 = .444. Now, $\boldsymbol{p}_0^0$ is temporarily split on input 0 into $\boldsymbol{p}_0^{0min}$ and $\boldsymbol{p}_0^{0max}$. $Min\{x_0 \ of \ t_0^0\}$ = 3.5 and $max\{x_0 \ of \ t_0^0\}$ = 5.0. Therefore, dividing T_0^0 results in T_0^{0min} containing

4.8	1.8	->	negative,
5.0	1.9	->	negative,
4.5	1.7	->	negative,

and T_0^{0max} containing

3.5	1.2	->	negative,
3.6	1.2	->	negative.

Temporarily recalculating prototypes gives

$\boldsymbol{p}_0^{0min}$=	4.767	1.800	
	0.205	0.082	-> negative,
$\boldsymbol{p}_0^{0max}$=	3.550	1.200	
	0.055	0.001	-> negative,
$\boldsymbol{p}_1^0$ =	3.650	1.300	
	0.450	0.158	-> positive.

The new table of distances now includes distances from all three prototypes for each instance:

		d_{0min}	d_{0max}	d_1
(4.8,	1.8)	0.161	622.73	5.720
(5.0,	1.9)	2.356	726.36	6.797
(4.5,	1.7)	2.522	517.27	4.421
(3.5,	1.2)	13.496	0.909	0.966
(3.6,	1.2)	13.001	0.909	0.744
(3.5,	1.2)	13.496	0.909	0.966
(3.0,	1.1)	17.156	110.00	2.710
(3.9,	1.4)	9.107	206.36	1.411
(4.2,	1.5)	6.424	311.82	2.488

Now, *T* is again classified, this time with the modified prototypes:

4.8	1.8	->	negative,	3.5	1.2	->	negative *
5.0	1.9	->	negative,	3.0	1.1	->	positive,
4.5	1.7	->	negative,	3.9	1.4	->	positive,
3.5	1.2	->	negative,	4.2	1.5	->	positive.
3.6	1.2	->	positive*,				

* misclassified

Now ε_a = 2/9 = .222. Since this is the first temporary split, ε_{small} is set equal to .222. $\boldsymbol{p}_0^{0min}$ and $\boldsymbol{p}_0^{0max}$ are now reunited into $\boldsymbol{p}_0^0$, and whole process is repeated for $\boldsymbol{p}_0^0$ on input 1, $\boldsymbol{p}_1^0$ on input 0 and $\boldsymbol{p}_1^0$ on input 1. Since ε_{small} is already less than ε_b, at least one permanent split will occur. If none of the other temporary splits classify better than 77.8% (7/9) correctly, $\boldsymbol{p}_0^0$ will be split on input 0.

3.2. Empirical Results. The data sets were reclassified using MSP in order to generate multiple prototypes per class. These results are shown in Table 2, which is an expansion of Table 1 that includes the new results.

Dataset	**SP**	**MSP**	**Best**	**Average**	**Worst**
Wine	94.6	**96.8**	93.8	92.6	90.5
Vowel	46.8	**81.1**	92.6	81.8	73.3
Sonar	73.3	**81.4**	78.9	75.7	72.5
Iris	95.0	**94.5**	95.3	94.7	94.0
Bupa	59.9	**62.1**	67.0	62.8	56.5
Pima	69.1	**70.3**	74.1	71.1	68.3
Vehicle	47.5	**68.1**	77.6	70.1	59.2
Thyroid	94.5	**95.8**	93.5	92.1	89.2
Glass	49.3	**63.4**	68.6	66.4	61.6
Waveform	79.5	**77.7**	72.0	68.7	66.0
Wavenoise	81.6	**77.1**	73.0	68.8	65.0

Table 2. Using multiple statistical prototypes to classify the data sets

As is to be expected, little improvement in performance was achieved for the data sets on which single prototypes already performed well. However, dramatic performance increases are evident on the data sets on which single prototypes performed poorly (i.e. glass, vowel, and vehicle).

4. Analysis

MSP outperforms all models on five of the data sets (sonar, thyroid, waveform, wavenoise, and wine) and performs near or above average on the others. Table 3 compares MSP with the other nine learning models by average performance over all eleven data sets. The models are ordered left to right in descending order of performance.

MSP	MML	Bayes	ID3	IB4	Ctree	IB3	Cart	Crules	CN2
78.9	78.0	77.8	77.7	77.6	76.8	76.7	75.5	74.2	71.3

Table 3. Average performance over all data sets

Table 4 shows the average number of prototypes created by MSP for each data set. The first column indicates the number of output classes for the appropriate data set. This may be thought of as a lower bound on the number of prototypes required by MSP. The second column indicates the average number of prototypes actually created by MSP. The third column indicates the number of instances in the training set (an upper bound on the number of prototypes), and the fourth column is the ratio #Prototypes/#Instances. This ratio gives an indication of parsimony, indicating to what degree the training information was able to be assimilated into a concise representation. As may be seen from the table, a high degree of parsimony is achieved in all cases. For all data sets, MSP created no more than an average of seven prototypes per output class, which always resulted in at least an 87% reduction in information stored.

Dataset	#Classes	#Prototypes	#Instances	%Ratio
Wine	3	5.8	125	4.6%
Vowel	11	48.1	370	13.0%
Sonar	2	14.7	146	10.1%
Iris	3	5.7	105	5.4%
Bupa	2	12.6	242	5.2%
Pima	2	10.7	538	2.0%
Vehicle	4	26.4	593	4.4%
Thyroid	3	6.0	1960	.3%
Glass	7	18.7	150	12.5%
Waveform	3	14.5	1400	1.0%
Wavenoise	3	16.2	1400	1.2%

Table 4. Number of prototypes created by MSP

4.1. Complexity Analysis. Time complexity for SP, which employs a single prototype per output class is reasonable. The calculation of all prototypes requires two passes over the data set. During the first pass, all m_{ij} are calculated. During the second pass, all σ_{ij} are calculated. Since there are n instances in the training set and each instance contains v input variables, the entire process requires $2nv$ steps which yields $O(nv)$.

MSP, which allows multiple prototypes per output class, initially requires the same $2nv$ steps as does SP since it actually first runs SP. Then each temporary split requires $O(n+nv)$ steps -- $O(n)$ steps to divide the instances between $\boldsymbol{p}_i^{kmin}$ and $\boldsymbol{p}_i^{kmax}$ and then $O(nv)$ to calculate the temporary means and standard deviations. During each pass through the “while” loop all prototypes are split in all dimensions. This means $O(nv)$ splits since the number of prototypes is bounded above by n, the number of instances. Therefore, a single pass through the while loop will require $O(nv(n+nv)) = O(n^2v+n^2v^2)) = O(n^2v^2)$. Finally, since each pass through the “while” loop yields a new prototype and since the number of prototypes is bounded above by n, no more than n passes through the loop will be made. Therefore, the entire algorithm is $O(n(n^2v^2)) = O(n^3v^2)$.

5. Conclusions

MSP, a prototype-based learning algorithm has been introduced. Empirical results show it to be robust in its generalization over a variety of problems as well as being parsimonious in its hypothesis representation.

Future research includes extending MSP to handle nominal data (some related work has been presented in [1][7][15]), investigating other metrics for prototype creation, and improving a parallel implementation based upon a *c-ary* tree presented in [17]. Parallel implementation of MSP using a *c*-ary tree with a broadcast and gather scheme is presented in [17].

This work was funded in part by a grant from Novell, Inc.

6. References

[1] Aha, D.W.; Kibler, D.; and Albert M.K, “Instance-Based Learning Algorithms”, *Machine Learning*, vol. 6, Kluwer Academic Publishers, The Netherlands, pp.37-66, 1991.

[2] Aha, D. W., “Incremental, instance-based learning of independent and graded concept descriptions”, *Proceedings of the Sixth International Workshop on Machine Learning*, pp. 387-391, 1989.

[3] Breiman, Leo; Friedman, Jorome; Olshen, Richard; and Stone, Charles, Classification and Regression Trees, Chapman and Hall, New York, 1993.

[4] Clark, Peter and Niblett, Tim, "The CN2 Induction Algorithm", *Machine Learning*, vol. 3, Kluwer Academic Publishers, The Netherlands, pp. 261-83, 1989.

[5] Dasarathy, Belur, Nearest Neighbor Norms: NN Pattern Classification Techniques, IEEE Computer Society Press, Los Alamitos, California, 1991.

[6] Gennari, John H.; Langley, Pat; and Fisher, Doug, "Models of Incremental Concept Formation", Machine Learning: Paradigms and Methods, ed. Jaime Carbonell, MIT press, Cambridge, Massachusetts, pp. 11-61, 1990.

[7] Giraud-Carrier, C. and Martinez, T., “An Efficient Metric for Heterogeneous Inductive Learning Applications in the Attribute-Value Language”, *Proceedings of the Third Golden West International Conference on Intelligent Systems*, 1994.

[8] Haykin, Simon, Neural Networks: A Comprehensive Foundation, Macmillan College Publishing Company, New York, 1994.

[9] Hecht-Nielsen, R., “Counterpropagation Networks”, *Applied Optics*, vol. 26, no. 23, pp. 4979-84, December, 1987.

[10] Quinlan, J. Ross, C4.5: Programs For Machine Learning, Morgan Kaufman Publishers, San Mateo, California, 1993.

[11] Quinlan, J. R., "Induction of Decision Trees", Readings in Machine Learning, eds. Jude W. Shavlik and Thomas G. Dietterich, Morgan Kaufman Publishers, Inc., San Mateo, California, pp. 57-69, 1990.

[12] Reilly, D. L.; Cooper, L. N.; Elbaum C., “Learning Systems based on Multiple Neural Networks”, (Internal paper), Nestor, Inc., 1988.

[13] Rissanen, Jorma, “A Universal Prior for Integers An Estimation by Minimum Description Length”, *Annal of Statistics*, vol. 11 no. 2, pp. 416-31, 1983.

[14] Rumelhart, D. E.; McClelland, J. L., Parallel Distributed Processing, MIT Press, chs. 5, 8, pp.151-193, 318-62, 1986.

[15] Stanfill, C. and Waltz, D., “Toward Memory-Based Reasoning”, *Communications of the ACM*, vol. 29, no. 12, pp. 1213-28, 1986.

[16] Tou, J. T., and Gonzalez, R., C., Pattern Recognition Principles, Addison-Wesley Publishing Company, Reading, Massachusetts, 1974.

[17] Ventura, Dan, “On Discretization as a Preprocessing Step for Supervised Learning Models”, Masters Thesis, Brigham Young University, April, 1995.

[18] Zarndt, Frederick, "An Empirically Derived Test Suite for Machine Learning and Connectionist Algorithms", in preparation, 1995.