Proceedings of the First New Zealand International Conference on Artificial Neural Networks and Expert Systems (ANNES), pp. 54-57, November 1993.

The Importance of Using Multiple Styles of Generalization*

D. Randall Wilson
e-mail: randy@axon.cs.byu.edu

Tony R. Martinez
e-mail: martinez@cs.byu.edu

Computer Science Department
Brigham Young University, Provo, Utah 84602, U.S.A.

Abstract

There are many ways for a learning system to generalize from training set data. There is likely no one style of generalization which will solve all problems better than any other style, for different styles will work better on some applications than others. This paper presents several styles of generalization and uses them to suggest that a collection of such styles can provide more accurate generalization than any one style by itself. Empirical results of generalizing on several real-world applications are given, and comparisons are made on the generalization accuracy of each style of generalization. The empirical results support the hypothesis that using multiple generalization styles can improve generalization accuracy.

1. Introduction

There are many ways for a computer to *learn by example*, and there are many styles of generalization to accomplish this task. There is likely no one style of generalization which will solve all problems, for different styles will work better on some applications than others. Eventually it should be possible to construct a system which can discover on its own which style or styles of generalization will work best on any given application. This paper presents several styles of generalization and suggests that a collection of such styles can provide more accurate generalization that any one by itself.

Traditional computer programming requires the programmer to explicitly define what the computer should do given any particular input. Even expert systems depend on the knowledge of experts in the field and their ability to explain how they make their decisions. When a solid knowledge of the application domain is available, this process has often worked quite well. In some applications, however, there is not enough known about the subject to derive a formula or write a program to solve the problem.

In such cases, it is often possible to collect a set of examples of what the output of the system should be in specific situations. Each such example is called an

* This research was supported in part by grants from Novell, Inc., and WordPerfect Corporation.

instance, and usually consists of a vector of input values (the *inputs*) representing certain features of the environment, as well as the value or values that the system should output (the *output*) in response to these input values. A collection of such instances is called a *training set*.

Much research has been done to come up with systems which learn by example. Given a training set of instances, these systems can often "learn" the relationship between the inputs and outputs well enough to be able to receive an input and produce the correct output with a high probability, even if that input was not one of the examples. This ability is called *generalization*.

There have been many styles of generalization employed in the areas of Neural networks [1,2,3] and machine learning, and each style has success on at least some applications.

The research presented in this paper makes use of several generalization styles in order to see if it is true that a collection of such styles will be able to generalize more accurately than any one style alone. These styles of generalization are presented in section 2.

Section 3 presents empirical results of simulations run on a variety of applications using each of these generalization styles, and gives a preliminary analysis of these results. Section 4 provides conclusions and future research areas.

2. Prototype Styles of Generalization

The research presented in this paper makes use of several styles of generalization, among which were several distance metrics, some critical variable detection algorithms, and voting schemes to enhance each of these. It is hypothesized that a collection of such styles will be able to provide more accurate generalization than any one style by itself.

These styles make use of *prototypes* and are simple enough that in many cases all of the styles can be tested against some or all of the instances in the training set to determine which style to use during generalization. *Prototypes* are similar to instances in the training set, but may represent several instances with the same output, and can have additional features. In this research prototypes consist of:

- A vector of input values representing features in the environment (*inputs*) and one or more output values, representing the supposedly correct response of the system to the input values (*output*).
- A *count* of how many instances the prototype represents,
- A count of how many instances with the same inputs had different outputs (called the *conflict*),
- A ratio of *count* to *total,* where *total* = *count* + *conflict*. This gives a probability that the inputs of the prototype should map to its output(s), and is called the *integrity*.

When prototypes are used, they are created during *learning*, and then compared with new inputs to determine what the output should be during *execution*. Figure 1 illustrates the information contained in a typical instance and prototype, respectively.

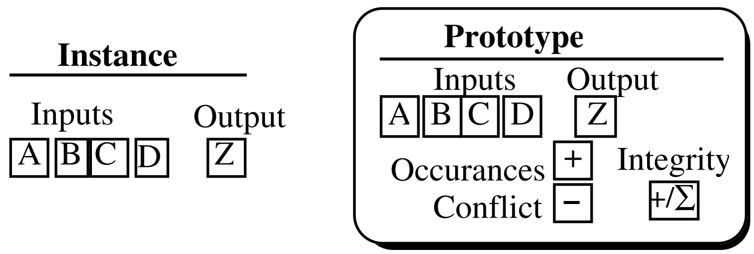

Figure 1. Instance vs. Prototype

An overview of the specific styles of prototype generalization used in this research is given in Figure 2, and the individual styles are discussed below.

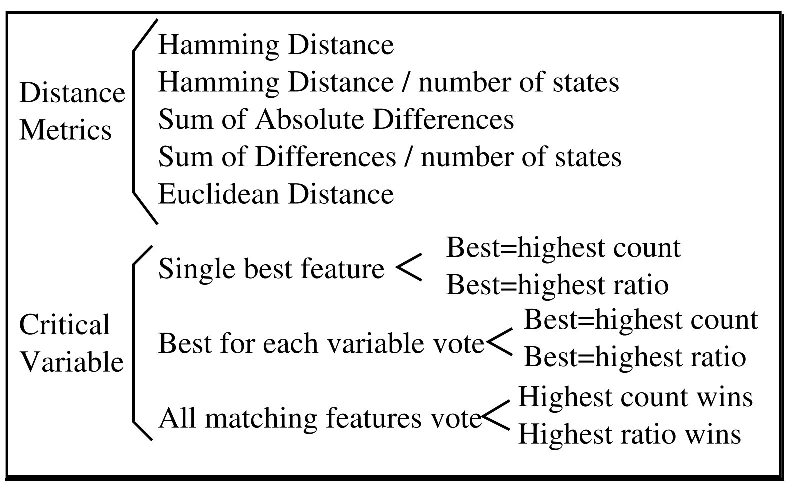

Figure 2. Generalization styles used.

2.2 Distance Metrics

Distance metrics are used to find out which prototype or prototypes are "closest" to the new input in question. The output value of the closest prototype or group of prototypes is used for the output value in response to the new input. There are several ways of determining how "close" an input is to one of the prototypes.

Hamming distance is an intuitive metric, and is simply the number of input variables which do not match the prototype. The prototype with the least number of mismatched variables wins using this metric.

In multi-state variables where the states represent a linear value, a sum of the (absolute) differences of the inputs of the prototype and the new inputs can be used as a measure of closeness.

An extension of the State Distance scheme is to normalize the distances by dividing the state difference for each variable by the number of states that that variable can have. For example, a difference of 1 for a boolean variable would be 1/2, while a difference of 1 for a 10-state variable would be 1/10. Similarly, the Hamming distance itself can be divided by the number of states, since being mismatched on a 10-state variable might be considered less "wrong" than being mismatched on a boolean one.

Euclidean distance is similar to the State Difference scheme, except that the individual distances are squared before they are summed. The State Difference and Euclidean Distance measures have been called the Minkowskian metric with r=1 and r=2, respectively [4].

Regardless of the distance metric used, there will often be several prototypes which are the same "distance" from the new input. Instead of arbitrarily choosing one of the several equally close prototypes to be the winner, the prototypes can vote for their output. One way to do this is to allow each prototype to cast a number of votes equal to its *count*, since each prototype may represent several instances.

Example 1. Consider a training set with four input variables, the first three of which are Boolean, and the last of which is a 10-state variable (with values 0 through 9). If a prototype with input values 0, 0, 1, and 5, respectively, was compared to the input 0, 1, 1, 1, the distance metrics mentioned above would be computed as shown in Figure 3.

Variable Name:	A	B	C	D		Out
Number of States:	2	2	2	10		2
Prototype:	**0**	**0**	**1**	**5**	→	**1**
New input to be classified:	**0**	**1**	**1**	**1**	→	**?**

Distance Metric	Distance
1. Hamming Distance:	0 + 1 + 0 + 1 = 2
2. Normalized Hamming Distance:	0 + 1/2 + 0 + 1/10 = 0.6
3. Absolute Difference:	0 + 1 + 0 + 4 = 5
4. Normalized Absolute Difference:	0 + 1/2 + 0 + 4/10 = 0.9
5. Euclidean Distance:	0 + 1 + 0 + 16 = 17

Figure 3. Results of various distance metrics (from example 1).

2.3 Critical Features

One way to determine which variables are most important is to find those which tend to have a strong correlation with an output value. In order to discover critical variables, a new kind of prototype, called a *first-order feature* is created for each input variable of each instance. Each feature has only the output and one input variable asserted, and the rest of the input variables are "don't care" variables (denoted by an asterisk, "*"). Such features are similar to rule-based instances with only one variable asserted [5].

It is possible to use combinations of variables instead of only one variable at a time and thus create higher-order features, but such methods run into exponential factors in both storage and computation time, and are thus not pursued in this research. The term *feature* will therefore be used to mean first-order feature.

Example 2. For the instance "0 0 1 5->1", the following four features would be generated:

0 * * * -> 1	Frequency: 1	Conflict: 0	Integrity: 1.0
* 0 * * -> 1	Frequency: 1	Conflict: 0	Integrity: 1.0
* * 1 * -> 1	Frequency: 1	Conflict: 0	Integrity: 1.0
* * * 5 -> 1	Frequency: 1	Conflict: 0	Integrity: 1.0

Figure 4. Features created from the instance "0 0 1 5->1".

If any of these features already existed in the system, the new one would be discarded, and the old one's "count" would be incremented and its integrity recomputed. If any features existed which had the same input but a different output (e.g., 0 * * * -> 0), then both features would increment their "conflict" and recompute their integrity. The first few instances in the training set will probably add quite a few features to the system, but later features will usually find matches for most of the features they try to add.

During generalization, a feature is said to *match* the input if the variable which is asserted matches the corresponding variable in the input. The other "don't care" variables are always considered to match.

There are several ways to use these prototypes for generalization. One is to select the one very best feature that matches the input and use its output. The "best" feature can be either the one with the highest count (meaning it occurred most often), or the one with the highest integrity (meaning it was relatively undisputed).

Another method is to select the best feature for each input variable, and let these vote on the output. A final method used in this research is to allow all matching features to vote for the output they want. In each case, a feature's voting power equals its *count*, which is the number of instances it represents.

3. Empirical Results

Each of the generalization styles described above was implemented and tested on several well-known training sets from the collection of Machine Learning Databases at University of California Irvine [6].

Out of *n* instances in each database, *n*-1 instances were used in the training set, and *n* instances were used in the test set, using a "leave-one-out" method. This was accomplished by building prototypes from the entire set of instances, and then temporarily "unlearning" each instance, one at a time. At that point it is as though the system had never seen the "unlearned" instance, so it can use the remaining prototypes to guess the output using each of the various generalization styles. Then each of the guessed output values is compared against the instance's real output to determine whether each style was correct. Finally, the instance is relearned, the next one is unlearned, and the process is repeated *n* times.

An overview of the results are given in Figure 5, above, which shows the accuracy of each generalization style for each application.

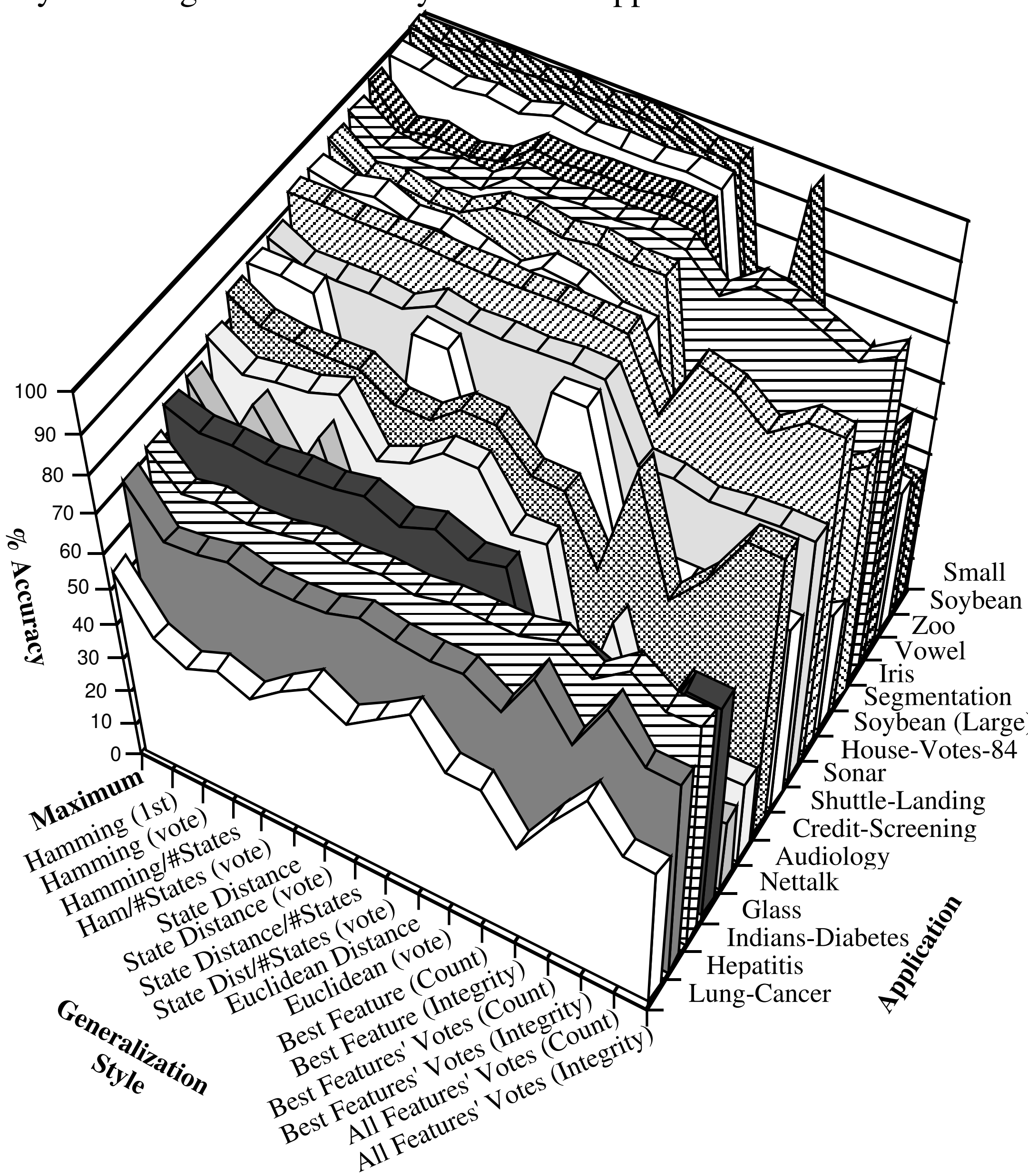

Figure 5. Overview of results.

The actual percentages are given in Table 6, below, along with a column labeled “Max” which gives the maximum generalization accuracy of any style for each database.

		Distance Metrics										**First-Order Features**					
	Max	Hamming Distance				State Distance				Euclid Dist.		Single best one		Best ones vote		All vote	
				/#st.				/#st.									
Database		1st	vt	1st	vt	1st	vt	1st	vt	1st	vt	cnt	int	cnt	int	cnt	int
Soybean (Small)	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	98	98	36	98	36	64	36	36
Zoo	**99**	98	97	98	97	**99**	98	98	98	**99**	98	41	78	41	62	41	41
Vowel	**98**	89	91	89	91	97	97	97	97	**98**	**98**	40	40	43	34	53	65
Iris	**96**	91	90	91	90	**96**	95	**96**	95	**96**	95	87	93	92	89	85	90
Segmentation	**95**	89	90	89	90	94	**95**	94	**95**	94	94	28	73	56	62	63	69
House-Votes-84	**93**	**93**	**93**	**93**	**93**	**93**	**93**	**93**	**93**	**93**	**93**	76	92	89	83	89	89
Soybean (Large)	**93**	91	**93**	90	91	89	87	92	91	87	87	00	58	02	46	29	35
Sonar	**88**	84	84	84	84	**88**	87	87	87	**88**	**88**	70	70	68	69	70	70
Shuttle-Landing	**87**	**87**	**87**	40	40	**87**	**87**	53	53	**87**	**87**	53	53	53	53	53	53
Credit-Screening	**86**	81	81	81	81	76	76	82	82	73	73	56	**86**	56	66	80	80
Audiology	**81**	75	77	78	**81**	68	71	78	78	65	66	24	53	24	44	24	24
Nettalk	**77**	63	**77**	63	**77**	48	59	48	59	42	47	17	42	17	40	19	21
Glass	**76**	73	74	73	74	74	**76**	73	**76**	72	74	37	58	55	44	63	63
Indians-Diabetes	**72**	64	65	64	65	67	66	67	66	68	68	68	71	65	**72**	66	66
Hepatitis	**70**	58	59	61	59	59	60	63	61	60	61	55	69	55	**70**	58	58
Lung-Cancer	**53**	38	34	38	34	41	47	41	47	**53**	41	41	28	41	**53**	41	41

Table 6. Percentages of Accurate Generalization

The results are encouraging, for they support the hypothesis that no one style will generalize best on all applications, and that it is useful to have a collection of generalization styles to choose from. Note, for example, that in the Hepatitis database (second from front in Figure 5) the distance metrics did not do as well as some of the feature methods. However, on others, such as the Zoo database, the distance metrics did much better.

In some applications there is at least one variable with a high correlation to the output value, allowing first-order feature methods to perform well. When there are also some very uncorrelated variables, distance metrics get thrown off, causing them to do less well. In other applications, on the other hand, first-order features have trouble because there are groups of variables which together determine the output. Among the distancemetrics themselves, voting generally helps accuracy except when the system happens to get lucky. State and Euclidean distance metrics work well on linear variables, but make little sense on nominal-valued variables.

Regardless of the specific reasons for the variations of accuracy among generalization styles, using several styles of generalization and picking the one which generalizes most accurately on each application provides greater overall generalization accuracy than any one style alone.

4. Future Research and Conclusions

The results of this research are promising, and support the hypothesis that using multiple styles of generalization can be more effective than using any one style by itself.

There are a variety of simple styles of generalization which can be added to this system's repertoire. Combining styles could be beneficial as well, such as using first-order features to weight the variables when calculating distances, and to weight voting capability of prototypes as well. Some success has recently been achieved in these areas.

Current research is continuing to more fully answer questions such as:

• What new styles of generalization can be added to the system?
• What extensions to these styles are useful?
• When did each fail, and why?
• How these styles can be combined?
• Which styles seem most useful and general overall?
• How can these algorithms be efficiently performed in a massively parallel architecture?

The final goal of this research is to construct systems which automatically determine the generalization style or styles that work best with any given application, and thus provide fast, accurate solutions to applications for which automated solutions were previously intractable or unknown.

Bibliography

[1] Lippmann, Richard P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, **3**, no. 4, pp. 4-22, April 1987.

[2] Rumelhart, D. E., and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, Ch. 8, pp. 318-362, 1986.

[3] Widrow, Bernard, and Michael A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," *Proceedings of the IEEE*, **78**, no. 9, pp. 1415-1441, September 1990.

[4] Duda, R. O., and P. E. Hart (1973). *Pattern Classification and Scene Analysis*. New York, NY: Wiley.

[5] Martinez, Tony R., "Adaptive Self-Organizing Concurrent Systems," *Progress in Neural Networks*, **1**, ch. 5, pp. 105-126, O. Omidvar (Ed), Ablex Publishing, 1990.

[6] Murphy, P., and D. W. Aha, *UCI Repository of Machine Learning Databases* [Machine-readable data repository at ics.uci.edu], Irvine, CA: University of California, Department of Information and Computer Science, 1993.