Advances in Instance-Based Learning Algorithms

A Dissertation
Presented to the
Department of Computer Science
Brigham Young University

In Partial Fulfillment
of the Requirement for the Degree
Doctor of Philosophy

by
D. Randall Wilson
August 1997

This dissertation by D. Randall Wilson is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the dissertation requirements for the degree of Doctor of Philosophy.

Tony R. Martinez, Committee Chairman

Gordon E. Stokes, Committee Member

Parris K. Egbert, Committee Member

Theodore A. Norman, Committee Member

J. Kelly Flanagan, Committee Member

5/29/97

Date

Scott N. Woodfield, Graduate Coordinator

Table of Contents

I. Background and Motivation **1**

1. Introduction **2**
1. Supervised Learning and Classification 2
2. Nearest Neighbor Algorithms 3
3. Thesis Statement 4
4. Overview of Dissertation 4
5. Publications 5

2. Related Work **8**
1. Bias and Generalization 8
2. Artificial Neural Networks 8
3. Other Learning Models 9
4. Nearest Neighbor / Instance-Based Algorithms 10
    4.1. Distance Functions 10
        4.1.1. Value Difference Metric 10
        4.1.2. Discretization 11
        4.1.3. Unknown Values 11
    4.2. Speeding Classification 11
    4.3. Reducing the Instance Set 11
        4.3.1. "Edited" Nearest Neighbor Classifiers 11
        4.3.2. "Instance-Based" Algorithms 12
        4.3.3. Prototypes and other modifications 12
    4.4. Weighting Techniques 13
        4.4.1. Vote Weighing 13
        4.4.2. Attribute Weighting 13
5. Author's Previous Work 14

3. Bias and the Probability of Generalization **20**
1. Introduction 20
2. Why One Bias Cannot be "Better" than Another 21
3. Why One Bias Can be "Better" than Another 23
    3.1. Functions are Not Equally Important 23
    3.2. Bias of simplicity 24
    3.3. Additional Information 25
4. Characteristics of Algorithms and Applications 26
    4.1. Characteristics of Applications 27
        4.1.1. Number of input attributes (dimensionality) 27
        4.1.2. Type of input attributes. 27
        4.1.3. Type of output. 27
        4.1.4. Noise. 27
        4.1.5. Irrelevant attributes 27
        4.1.6. Shape of decision surface. 28
        4.1.7. Data density 28
        4.1.8. Order of attribute-class correlations. 28
    4.2. Characteristics of Learning Algorithms 28
5. Conclusions 29

II. Heterogeneous Distance Functions..32

4. Heterogeneous Radial Basis Function Networks..........................33
1. Introduction..33
2. Radial Basis Function Network 34
3. Heterogeneous Distance Function 35
4. Empirical Results...37
5. Conclusions & Future Research...39

5. Improved Heterogeneous Distance Functions 41
1. Introduction..41
2. Previous Distance Functions 43
   2.1. Normalization 44
   2.2. Attribute Types 45
   2.3. Heterogeneous Overlap-Euclidean Metric (HOEM)...........................45
   2.4. Value Difference Metric (VDM)..46
   2.5. Discretization 48
3. Heterogeneous Value Difference Metric (HVDM)....................................48
   3.1. Normalization 49
   3.2. Normalization Experiments...50
   3.3. Empirical Results of HVDM vs. Euclidean and HOEM.........................53
4. Interpolated Value Difference Metric (IVDM)...55
   4.1. IVDM Learning Algorithm 55
   4.2. IVDM and DVDM Generalization 57
5. Windowed Value Difference Metric (WVDM)..61
6. Empirical Comparisons and Analysis of Distance Functions 65
   6.1. Effects of Sparse Data...68
   6.2. Efficiency Considerations 69
      6.2.1. Storage 69
      6.2.2. Learning Speed 70
      6.2.3. Generalization Speed 70
7. Related Work 70
8. Conclusions & Future Research Areas..72
Appendix. Handling Skewed Class Distributions with HVDM..................73

III. Instance Set Reduction..79

6. Improved Center Point Selection for Probabilistic Neural Networks...80
1. Introduction..80
2. Reduction Algorithm 82
3. Empirical Results...84
4. Conclusion 85

7. Reduction Techniques for Exemplar-Based Learning Algorithms.......87
1. Introduction..87
2. Issues in Exemplar Set Reduction...89
   2.1. Representation...89
   2.2. Direction of Search...89

2.2.1. Incremental...89
2.2.2. Decremental...90
2.2.3. Batch...90
2.3. Border points vs. central points...91
2.4. Distance Function...91
2.5. Voting...93
2.6. Evaluation Strategies...94
2.6.1. Storage reduction...94
2.6.2. Speed increase...94
2.6.3. Generalization accuracy...94
2.6.4. Noise tolerance...94
2.6.5. Learning speed...95
2.6.6. Incremental...95
3. Survey of Instance Reduction Algorithms...95
3.1. Nearest Neighbor Editing Rules...96
3.1.1. Condensed Nearest Neighbor Rule...96
3.1.2. Selective Nearest Neighbor Rule...96
3.1.3. Reduced Nearest Neighbor Rule...97
3.1.4. Edited Nearest Neighbor Rule...97
3.1.5. All-kNN...98
3.1.6. Variable Similarity Metric...98
3.2. "Instance-Based" Learning Algorithms...98
3.2.1. IB2...98
3.2.2. Shrink (Subtractive) Algorithm...99
3.2.3. IB3...99
3.2.4. IB4 and IB5...100
3.2.5. Model Class Selection...100
3.2.6. Typical Instance-Based Learning...100
3.2.7. Random Mutation Hill Climbing...101
3.2.8. Encoding Length...101
3.3. Prototypes and Other Modifications of the Instances...102
3.3.1. Prototypes...102
3.3.2. RISE...102
3.3.3. EACH...102
4. New Reduction Algorithms: DROP1-5 and DEL...103
4.1. DROP1...103
4.2. DROP2: Using More Information and Ordering the Removal...104
4.3. DROP3: Filtering Noise....105
4.4. DROP4: More Carefully Filtering Noise...105
4.5. DROP5: Smoothing the Decision Boundary...105
4.6. Decremental Encoding Length...106
5. Experimental Results...106
5.1. Results...106
5.2. Effect of Noise...108
6. Conclusions and Future Research Directions...110

IV. Attribute and Vote Weighting 115

8. Instance-Based Learning with Genetically Derived Attribute Weights **116**
1. Introduction 116
2. Distance Function 117
    2.1. Normalization 117
    2.2. Heterogeneous Distance Function 117
    2.3. Irrelevant Attributes 119
    2.4. Redundant Attributes 119
3. Evolutionary Algorithm 120
    3.1. Genetic Operators 121
    3.2. Parent Selection 121
    3.3. Evaluation Function 123
4. Experiments 124
    4.1. Testing Irrelevant Attributes 125
    4.2. Testing Redundant Attributes 126
5. Conclusions & Future Research 126

9. Distance-Weighting and Confidence in Instance-Based Learning **128**
1. Introduction 128
2. Fuzzy Instance-Based Learning (FIBL) Algorithm 130
    2.1. Vote Weighting 130
    2.2. Heterogeneous Distance Function 132
3. Cross-Validation and Confidence (CVC) 133
    3.1. Cross-Validation 133
    3.2. Confidence 134
    3.3. Cross-Validation and Confidence (CVC) 134
4. FIBL Learning Algorithm 135
5. Experimental Results 137
6. Related Work 139
7. Conclusions and Future Research Directions 140

V. Conclusion 142

10. Advances in Instance-Based Learning **143**
1. Introduction 143
2. Heterogeneous Distance Functions 144
    2.1. Linear Distance Functions 145
    2.2. Value Difference Metric for Nominal Attributes 145
    2.3. Interpolated Value Difference Metric 147
3. Instance Pruning Techniques 149
    3.1. Speeding Classification 149
    3.2. Reduction Techniques 149
    3.3. DROP4 Reduction Algorithm 150
4. Distance-Weighting and Confidence Levels 152
    4.1. Distance-Weighted Voting 153

4.2. Cross-Validation and Confidence (CVC) 155
4.2.1. Cross-Validation 155
4.2.2. Confidence 155
4.2.3. Cross-Validation and Confidence (CVC) 156
4.3. Parameter Tuning 157
5. IDIBL Learning Algorithm 159
6. Empirical Results 161
7. Conclusions and Future Research 164

11. Conclusions & Future Research Directions 169
1. Summary and Contributions 169
2. Future Research Directions 170

Abstract 172

Acknowledgments

Thanks to my advisor Dr. Tony Martinez for his unending assistance in this work. He taught me how to do quality research and how to publish new findings in refereed conferences and journals. He has also been my advocate in obtaining funding that has supported my family while I finished my graduate work.

Thanks also go to many other faculty members who contributed to my education through their teaching efforts and individual guidance.

Several machine learning researchers have contributed significantly to my research as well, including David Aha, who provided numerous helpful references and suggestions; Mike Cameron-Jones, who recently provided guidance and source code for algorithms related to my work; and several anonymous reviewers who gave useful suggestions for several of my papers.

I would also like to thank Novell, Inc.; WordPerfect Corp.; and especially the Brigham Young University Computer Science Department for financial support.

I am especially grateful to my wife and best friend, Linette, who has been a wonderful strength to me throughout my graduate work. Her love and support have made life wonderful for me. I am also grateful for my cute kids who bring such joy to my life.

Finally, I am ever in debt to my Heavenly Father, who has blessed me beyond explanation. I give Him credit for any truly valuable ideas presented in this dissertation. I hope that I can use the knowledge and experience gained through this education to serve Him and bless the lives of others.

Part I

Background and Motivation

"In the beginning...."—Genesis 1:1

Part I provides the background information and motivation for the remainder of the dissertation.

Chapter 1 provides an introduction to the research contained in this dissertation. It introduces inductive learning, generalization, and the nearest neighbor rule, including its advantages and disadvantages. It then outlines the remainder of the dissertation.

Chapter 2 presents related work. It discusses theoretical work relating to generalization and bias and lists a variety of models that have been used for inductive learning, including neural networks, genetic algorithms, and machine learning algorithms. It then describes previous research in instance-based learning, including distance metrics, reduction techniques, hybrid models, and weighting schemes.

Chapter 3 discusses arguments that have been made regarding the impossibility of any learning algorithm or *bias* achieving higher generalization accuracy than any other. It shows that it is possible to improve generalization accuracy in practice, and gives several suggestions on how this can be done. Chapter 3 has been submitted for review and may be referenced as follows.

Wilson, D. Randall, and Tony R. Martinez, (1997). "Bias and the Probability of Generalization," submitted to *International Conference on Intelligent Information Systems (IIS'97).*

Chapter 1

Introduction

"Hey— 'Y' starts with 'W'. . . and 'W' starts with 'D'!"
—Adam Wilson, age 3, upon seeing "Y" Mountain.

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. Machine learning methods attempt to give machines the ability to learn from examples so that they can attain high accuracy at a low cost.

Humans are able to solve a wide variety of problems well, even in the face of incomplete domain knowledge. Humans use trial-and-error, past experience, heuristic rules and common sense to adapt to new situations. A computer program is typically much more rigid, because a programmer must determine in advance how the program should respond to any given set of inputs. However, unless the domain knowledge is sufficiently developed, it may be impossible or impractical for a programmer to know beforehand what rules to use in deciding upon an appropriate response to each possible situation.

In such cases, machine learning algorithms can be used as a tool to make reasonable solutions possible or good solutions more economical. A machine learning solution is often more accurate than a hard-coded program because it learns from actual data instead of making assumptions about the problem. It can often adapt as the domain changes and may take less time to find a good solution than would a programmer. In addition, machine learning solutions can typically work well even in the face of unforeseen circumstances.

1. Supervised Learning and Classification

Machine learning algorithms can use either *supervised* or *unsupervised* learning. This dissertation will focus on supervised learning systems and especially on classification.

In *supervised learning*, a system receives feedback to help it learn. The most common form of feedback used by a supervised learning system is a *training set*, which is a set of examples (or *instances*) that have both an input vector and an output value. In *classification*, the output value is a discrete *class*. During the learning phase, the machine internalizes the examples and attempts to learn how to map input vectors into the correct output class.

During execution, the machine is given new input vectors, most of which it has never encountered before. It must predict the correct class based on the preclassified examples it has already seen. This process is called *generalization*. Generalization accuracy is typically measured by using a *test set*, which contains instances with known output classes but which the algorithm does not see during the learning phase. The algorithm attempts to classify each instance in the test set, and its prediction is compared with the actual class of each instance to see if it generalized correctly.

A wide variety of learning models are available for application to supervised learning problems such as neural networks, decision trees, rule-based systems, and genetic algorithms. Each of these models has its own strengths and weaknesses. For example, many neural network learning models require a large number of iterations through the training set, which can make them impractical for use on some large or difficult problems. Learning algorithms also may have system parameters that must be selected by the user, which prevents the system from being fully automatic. Such parameters can have a large effect on the performance of each model, and often there is little guidance as to how these parameters should be selected. Some algorithms can also get permanently stuck in *local minima* without reaching a reasonable solution and must be restarted in such cases.

2. Nearest Neighbor Algorithms

The *Nearest Neighbor* algorithm [Cover & Hart, 1967] is a supervised learning algorithm that simply retains the entire training set during learning. During execution, the new input vector is compared to each instance in the training set. The class of the instance that is most similar to the new vector (using some distance function) is used as the predicted output class.

The nearest neighbor algorithm has several strengths when compared to most other learning models:

- It learns very quickly (O(n) for a training set of n instances).
- It is guaranteed to learn a consistent training set (i.e., one in which there are no instances with the same input vector and different outputs) and will not get stuck in local minima.
- It is intuitive and easy to understand, which facilitates implementation and modification.
- It provides good generalization accuracy on many applications. For example, see [Fogarty, 1992].

However, in its basic form the nearest neighbor algorithm has several drawbacks:

- Its distance functions are typically inappropriate for applications with both linear and nominal attributes.
- It has large storage requirements, because it stores all of the available training data in the model.
- It is slow during execution, because all of the training instances must be searched in order to classify each new input vector.
- Its accuracy degrades with the introduction of noise.
- Its accuracy degrades with the introduction of irrelevant attributes.
- It has no means of adjusting its decision boundaries after storing the data.

Many researchers have developed extensions of the nearest neighbor algorithm, which are commonly called *instance-based* [Aha, Kibler & Albert, 1991] learning algorithms. However, previous models have remained weak in at least some of the above areas.

We recognized the power of the basic nearest neighbor rule and saw potential for even better performance. The research reported in this dissertation has thus been directed at extending existing approaches in order to overcome each of the weaknesses listed above. It introduces new heterogeneous distance functions that significantly improve generalization accuracy for applications that have both linear and nominal attributes. It presents new techniques for reducing the number of instances stored in the classification system, which also speeds up classification and can make the system less sensitive to noise. It also introduces an attribute-weighting scheme that reduces sensitivity to irrelevant attributes. In addition, it proposes methods for using weighting schemes and confidence levels to fine-tune the classifier in order to improve generalization accuracy in practice.

3. Thesis Statement

Instance-based learning algorithms have had success on many applications, and yet they often suffer from large storage requirements, slow classification, sensitivity to noise and irrelevant attributes, inflexibility, and a lack of appropriate distance functions. This dissertation introduces new instance-based learning algorithms which improve on existing nearest neighbor techniques in each of these areas. Improved heterogeneous distance functions are introduced, instance reduction techniques are proposed, and weighting schemes are used to provide a flexible model capable of adapting to individual problems. The *Integrated Decremental Instance-Based Learning Algorithm* combines the advantages of each of these improvements into a comprehensive system.

4. Overview of Dissertation

Part I of this dissertation consists of three introductory chapters, including this one. Chapter 2 presents a discussion of related work in the area of inductive learning systems in general and instance-based learning in particular, as well as in distance functions and weighting schemes for instance-based learning algorithms. A list of instance reduction techniques is also given, but the details for related work in this area is deferred to Chapter 7 where an in-depth survey is presented.

The remainder of this dissertation (with the exception of the final chapter) consists of a collection of papers that have been either published or submitted for publication in journals or conference proceedings. The references for these papers are listed in the following section of this introduction and also appear at the beginning of the chapter to which each applies.

Chapter 3 serves an introductory role by discussing the ability of learning algorithms to generalize. In particular, arguments have been made in recent years to suggest that no learning algorithm can be "better" than any other in terms of generalization accuracy. Chapter 3 gives examples illustrating these arguments in the theoretical case but also shows how one algorithm *can* be considered better than another in practice and indicates how average generalization accuracy can be improved in practice.

With this established, Part II presents two chapters that introduce distance functions which result in improved generalization (on average) for applications with both nominal and linear attributes. Chapter 4 presents the *Heterogeneous Radial Basis Function* (HRBF) neural network, which is a probabilistic neural network [Specht, 1992] using a heterogeneous distance function. The heterogeneous distance function uses Euclidean distance for linear attributes and a derivative of the *Value Difference Metric* (VDM) [Stanfill & Waltz, 1986] for nominal attributes. Chapter 5 extends the distance function presented in chapter 4 and presents two extensions of the VDM that allow it to be used directly on both nominal and linear attributes. These distance functions are compared with each other and shown to result in significantly higher average generalization accuracy than previously used distance functions.

Part III presents two chapters that introduce algorithms for reducing storage and improving classification speed while maintaining good generalization accuracy and reducing sensitivity to noise. Chapter 6 presents the *Reduced Probabilistic Neural Network* (RPNN) that removes nodes from a traditional PNN and also uses the heterogeneous distance function first used by the HRBF presented in Chapter 4. Chapter 7 presents a survey of many instance reduction techniques used in conjunction with nearest neighbor classifiers and their derivatives. Several new algorithms are also presented that show improved generalization accuracy over the other methods surveyed and show better reduction than many of them.

Part IV presents two chapters on using various weighting schemes and fine-tuning parameters of the system. Chapter 8 introduces the *Genetic Instance-Based Learning* (GIBL) algorithm, which uses a genetic algorithm to find attribute weights on a basic nearest neighbor algorithm in order to reduce the sensitivity of the system to irrelevant attributes.

Chapter 9 introduces a distance-weighted instance-based learning system called the *Fuzzy Instance-Based Learning* (FIBL) algorithm that uses confidence levels in conjunction with cross-validation to choose good parameter settings in order to more accurately generalize.

Part V presents two chapters that conclude the dissertation. Chapter 10 presents a comprehensive system called the *Integrated Decremental Instance-Based Learning* (IDIBL) algorithm that combines the successful elements of previous chapters. It gives empirical results that indicate that IDIBL yields improved accuracy over any of the previous systems presented in this dissertation. IDIBL is also compared with results reported for a variety of well-known machine learning and neural network models. In these comparisons IDIBL achieves the highest average generalization accuracy of any of the models considered. The final chapter presents conclusions and directions for future research.

5. Publications

Chapters 3-10 are based on a collection of papers that have been either published or submitted for publication in refereed journals or conferences. Following is a list of references for these publications in the order in which they appear in this dissertation.

I. Background and Motivation

Wilson, D. Randall, and Tony R. Martinez, (1997). "Bias and the Probability of Generalization", submitted to *International Conference on Intelligent Information Systems (IIS'97)*. (Chapter 3).

II. Heterogeneous Distance Functions

Wilson, D. Randall, and Tony R. Martinez, (1996). "Heterogeneous Radial Basis Functions," *Proceedings of the International Conference on Neural Networks (ICNN'96)*, vol. 2, pp. 1263-1267, June 1996. (Chapter 4).

Wilson, D. Randall, and Tony R. Martinez, (1996). "Value Difference Metrics for Continuously Valued Attributes," *International Conference on Artificial Intelligence, Expert Systems and Neural Networks (AIE'96),* pp. 74-78. (Chapter 5).

Wilson, D. Randall, and Tony R. Martinez, (1997). "Heterogeneous Distance Functions for Instance-Based Learning," *Journal of Artificial Intelligence Research (JAIR)*, vol. 6, no. 1, pp. 1-34. (Chapter 5).

III. Instance Set Reduction

Wilson, D. Randall, and Tony R. Martinez, (1997). "Improved Center Point Selection for Radial Basis Function Networks," In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97)*. (Chapter 6).

Wilson, D. Randall, and Tony R. Martinez, (1997). "Instance Pruning Techniques," To appear in Fisher, D., ed., *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, Morgan Kaufmann Publishers, San Francisco, CA. (Chapter 7).

Wilson, D. Randall, and Tony R. Martinez, (1997). "Reduction Techniques for Exemplar-Based Learning Algorithms," submitted to *Machine Learning Journal*. (Chapter 7).

IV. Attribute and Vote Weighting

Wilson, D. Randall, and Tony R. Martinez, (1996). "Instance-Based Learning with Genetically Derived Attribute Weights," *International Conference on Artificial Intelligence, Expert Systems and Neural Networks (AIE'96),* pp. 11-14. (Chapter 8).

Wilson, D. Randall, and Tony R. Martinez, (1997). "Distance-Weighting and Confidence in Instance-Based Learning," submitted to *Computational Intelligence*. (Chapter 9).

V. Conclusion

Wilson, D. Randall, and Tony R. Martinez, (1997). "Advances in Instance-Based Learning," submitted to *Machine Learning Journal*. (Chapter 10).

The reference for each of these publications appears at the beginning of the chapter that includes the publication. In addition, the list of other publications mentioned or referenced within each chapter appears at the end of each chapter. Following are the papers referenced in Chapter 1.

References

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

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.

Fogarty, Terence C., (1992). "First Nearest Neighbor Classification on Frey and Slate's Letter Recognition Problem," *Machine Learning*, vol. 9, no. 4, pp. 387-388.

Rumelhart, D. E., and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, 1986.

Specht, Donald F., (1992). "Enhancements to Probabilistic Neural Networks," in *Proceedings International Joint Conference on Neural Networks (IJCNN '92)*, vol. 1, pp. 761-768.

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

Chapter 2

Related Work

"Well, he was humming this hum to himself, wondering what everybody else was doing, and what it felt like, being somebody else..."
—A. A. Milne, *Pooh Goes Visiting.*

The research presented in this dissertation would not have been possible without the groundwork laid by previous researchers in a variety of fields. This chapter briefly surveys work previously done in the areas addressed by this dissertation. It starts with a discussion of theoretical work regarding bias and generalization. It then lists several models that have been used for inductive learning. Finally, it discusses in more detail previous research on nearest neighbor algorithms, including work done on distance functions, pruning techniques, weighting methods, and alternate distance-based learning models. A more detailed treatment of related work is presented throughout the chapters of this dissertation.

1. Bias and Generalization

In order to generalize to previously unseen input vectors, an inductive learning algorithm must have a bias, which Mitchell [1980] defined as "any basis for choosing one generalization over another other than strict consistency with the observed training instances." Schaffer [1994] presented a *Conservation Law for Generalization Performance*, which argued that no bias could result in higher generalization accuracy than any other (including random guessing) when summed over all possible problems. He also showed that even using cross-validation to choose the apparently best learning algorithm for a problem does no better than random (over all problems), since it is itself simply another bias [Schaffer, 1993].

Wolpert [1993] provided proofs that Schaffer's *Conservation Law* findings are true but cautioned that the results must be interpreted with caution because learning algorithms can differ in their performance when the probability of some functions occurring is greater than others (which, in practice, is the case).

Aha [1992] made a strong case for the importance of determining *under what conditions* one algorithm outperforms another so that appropriate learning algorithms can be applied to solve problems.

2. Artificial Neural Networks

A variety of inductive learning algorithms exist to perform generalization. Artificial neural networks [Lippmann, 1987; Martinez, 1989; Wasserman, 1993; Widrow & Winter, 1988; Windrow & Lehr, 1990] use learning models inspired at least partially by our understanding of the operation of biological neural networks in brains. They typically use highly interconnected simple processing nodes.

The *backpropagation* neural network [Rumelhart & McClelland, 1986] is one of the most popular and successful neural network models. It uses a gradient descent method to learn weight settings that reduce error on the training set and has had much empirical success on a variety of real-world applications. However, it requires several user-defined parameters that can have a dramatic impact on its ability to solve the problem. It learns very slowly and can sometimes get stuck in *local minima* that keep it from ever reaching a good solution, which often requires the algorithm to start over. Recent advances in neural networks have helped to automate selection of some of the parameters and speed training [Fahlman, 1988, 1990; Buntine & Weigend, 1993; Møller, 1993].

Radial Basis Function (RBF) *Networks* [Broomhead & Lowe, 1988; Chen, 1991; Renals & Rohwer, 1989; Wong, 1991] are similar to the nearest neighbor classifier in that they have nodes with center points similar to prototypes in nearest neighbor systems. Distance functions are used to determine the activation at each node, and RBF's can be used to perform classification, as is done by a particular class of RBF networks called *Probabilistic Neural Networks* (PNN) [Specht, 1992]. Two chapters in this dissertation (Chapters 4 and 6) use PNN's rather than a nearest neighbor model to demonstrate the usefulness of advances in distance-based learning.

Other neural network (or *connectionist*) models include *Adaptive Resonance Theory* (ART) [Carpenter & Grossberg, 1987], *counterpropagation networks* [Hecht-Nielsen, 1987], the *self-organizing map* [Kohonen, 1990], and *ASOCS* [Martinez, 1986, 1990; Martinez & Vidal, 1988; Martinez & Campbell, 1991a, 1991b; Martinez, Campbell & Hughes, 1994; Rudolph & Martinez, 1991].

For more information on artificial neural networks in general, the reader is referred to the introductory article by Lippman [1987], or to the text by Wasserman [1993].

3. Other Learning Models

A variety of approaches to inductive learning have also been developed in the machine learning and artificial intelligence communities. *Decision tree* methods such as ID3 [Quinlan, 1986] and C4.5 [Quinlan, 1993] have had success in many domains. These models use information gain to determine which attribute to perform a split on (and where to split) and continue splitting until most of the instances can be classified correctly. Often this process is followed by a pruning step in which leaf nodes are pruned in order to avoid noise and to prevent *overfitting*, i.e., to avoid going beyond the underlying function and learning the sampling distribution.

Rule-based learning algorithms have also been developed, including AQ [Michalski, 1969], CN2 [Clark & Niblett, 1989] and RISE [Domingos, 1995].

For a more detailed information on machine learning in general, the reader is referred to [Mitchell, 1997].

Genetic Algorithms [Kelly & Davis, 1991; Spears et al., 1993] are search heuristics based on genetic operators such as crossover and mutation. They are sometimes used in conjunction with neural networks or machine learning algorithms to optimize parameters, and are sometimes used to attack problems directly. In this dissertation genetic algorithms are used to find attribute weights in the *Genetic Instance-Based Learning* (GIBL) algorithm in Chapter 8.

4. Nearest Neighbor / Instance-Based Algorithms

Cover & Hart [1967] formally introduced the nearest-neighbor rule and showed that its error rate is bounded by twice the Bayes optimum in the limit. Their original paper also mentions the *k*NN rule, in which the majority class of the *k* closest neighbors is used for classification. This reduces susceptibility to noise in some cases but can also result in lower accuracy in others. Also, the user often has to pick the value of *k*.

Salzberg et al. [1995] found upper and lower bounds on the number of examples it would take to learn a concept if the instances were chosen in the best way possible. They showed that some problems need exponential numbers of examples and are thus inherently difficult for the nearest-neighbor algorithm. They also identified some geometric properties that make some problems inherently easy.

Dasarathy [1991] surveys much of the early work done in the nearest-neighbor arena, and the reader is referred there for many of the original papers in this area.

4.1. Distance Functions

The original nearest neighbor paper [Cover & Hart, 1967] states that any distance function can be used to determine how “close” one instance is to another, but only Euclidean distance was used in their experiments. Euclidean distance is by far the most widely used distance function in instance-based learning systems. However, a variety of other distance functions are also available, including the Minkowskian distance metric [Batchelor, 1978; Duda & Hart, 1973] (which includes Euclidean distance as a special case), Mahalanobis distance [Nadler & Smith, 1993], context-similarity distance [Biberman, 1994], and many others [Diday, 1974; Michalski, Stepp & Diday, 1981].

One problem with most of these distance functions is that they are not designed for *nominal* attributes, i.e., those with discrete, unordered input values (such as *color*={*red, green, blue, brown*}). In such cases using linear distance makes little sense. Giraud-Carrier & Martinez [1995] introduced a *heterogeneous* distance function that uses linear distance on linear attributes and the overlap metric on nominal attributes (where the *overlap* metric gives a distance of zero if the two values are equal and a distance of one otherwise). A similar function is used by many of Aha’s instance-based learning algorithms [Aha, Kibler & Albert, 1991; Aha, 1992].

4.1.1. Value Difference Metric

Stanfill & Waltz [1986] introduced the *Value Difference Metric* (VDM), which finds a real-valued distance between nominal attribute values based on statistics gathered from the training set. This improves classification over the simple overlap metric on applications with nominal attributes. Cost & Salzberg [1993] modify this metric and Rachlin et al. [1994] report on using this *Modified VDM* (MVDM) in a classification system called PEBLS.

This dissertation presents several heterogeneous distance functions that extend the VDM to handle linear attributes as well as nominal attributes.

4.1.2. Discretization

Some systems, such as PEBLS [Rachlin et al., 1994], ID3 [Quinlan, 1986], and ASOCS [Martinez, 1990] cannot directly use continuously valued input attributes, but must first preprocess such attributes into discrete values, a process called *discretization*. Several researchers have looked at how best to perform discretization [Ventura & Martinez, 1995; Ventura, 1995; Lebowitz, 1985]. Ting [1994, 1996] found discretization to improve performance in some cases, though the reasons for this may have more to do with skewed class distributions than discretization (see Chapter 5, Section 7 for discussion on this point).

The algorithms presented in this dissertation to not use discretization but comparisons are made with systems that do, especially in Chapter 5.

4.1.3. Unknown Values

Many datasets contain unknown attribute values (i.e., "?" instead of a real value) that must be handled by a practical system. Dixon [1979] gives six methods of handling unknown values, Quinlan [1989] presents several more, and Aha [1991] and Giraud-Carrier & Martinez [1995] both assume that the distance to an unknown value is maximal.

4.2. Speeding Classification

As mentioned in the introduction in Chapter 1, one of the disadvantages to the nearest neighbor algorithm is the time required to classify new input vectors. The basic algorithm works by finding the distance between the new input vector and every instance in the training set in order to find the nearest neighbor(s).

It is possible to use k-dimensional trees ("k-d trees") [Wess, Althoff & Derwand, 1994; Sproull, 1991; Deng & Moore, 1995] to find the nearest neighbor in O(log$n$) time in the best case. However, as the dimensionality grows, the search time can degrade to that of the basic nearest neighbor rule.

Another technique used to speed the search is *projection* [Papadimitriou & Bentley, 1980], where instances are sorted once by each dimension, and new instances are classified by searching outward along each dimension until it can be sure the nearest neighbor has been found.

4.3. Reducing the Instance Set

One of the most straightforward ways to speed classification in a nearest-neighbor system is by removing some of the instances from the instance set. This also addresses another of the main disadvantages of nearest-neighbor classifiers—their large storage requirements. In addition, it is sometimes possible to remove noisy instances from the instance set and actually improve generalization accuracy.

Several techniques for reducing storage are listed here, but more details are provided in Chapter 7, where an in-depth survey is presented.

4.3.1. "Edited" Nearest Neighbor Classifiers

Hart [1968] made one of the first attempts to reduce the size of the training set with his *Condensed Nearest Neighbor Rule* (CNN), and Ritter et al. [1975] extended the CNN in their *Selective Nearest Neighbor Rule* (SNN). These algorithms are both extremely sensitive to noise.

Gates [1972] modified the CNN algorithm in his *Reduced Nearest Neighbor Rule* (RNN), which is able to remove noisy instances and internal instances while retaining border points. Swonger [1972] presented the *Iterative Condensation Algorithm* (ICA) that allows both addition and deletion of instances from the instance set.

Wilson [1972] developed an algorithm which removes instances that do not agree with the majority of their k nearest neighbors (with k=3, typically). This edits out noisy instances as well as close border cases, leaving smoother decision boundaries, but it also retains all internal points, which keeps it from reducing the storage requirements as much as most other algorithms. Tomek [1976] extended Wilson's algorithm with his *All k-NN* method of editing by calling Wilson's algorithm repeatedly with k=1..k, though this still retains internal points.

4.3.2. "Instance-Based" Algorithms

Aha et. al. [1991] presented a series of *instance-based* learning algorithms that reduce storage. *IB2* is quite similar to the Condensed Nearest Neighbor (CNN) rule [Hart, 1968] and suffers from the same sensitivity to noise. *IB3* [Aha et al. 1991] addresses IB2's problem of keeping noisy instances by using a statistical test to retain only *acceptable* misclassified instances. In their experiments, IB3—due to its reduced sensitivity to noise—was able to achieve greater reduction in the number of instances stored and also achieved higher accuracy than IB2 on the applications on which it was tested.

Zhang [1992] used a different approach called the *Typical Instance Based Learning* (TIBL) algorithm, which attempted to save instances near the center of clusters rather than on the border.

Cameron-Jones [1995] used an *encoding length heuristic* to determine how good a particular subset is in modeling the entire training set. After a growing phase similar to IB3, a random mutation hill climbing method called *Explore* is used to search for a better subset of instances using the encoding length heuristic as a guide. It achieves by far the greatest reduction in storage of all of the above methods as well as providing reasonable accuracy.

4.3.3. Prototypes and other modifications

Instead of just deciding which instances in the original training set to keep, some algorithms seek to reduce storage requirements and speed up classification by modifying the instances themselves.

Chang [1974] introduced an algorithm in which each instance in the original training set is initially treated as a *prototype*. The nearest two instances that have the same class are merged into a single prototype that is located somewhere between the two prototypes. This process is repeated until classification accuracy starts to suffer.

Domingos [1995] introduced the RISE 2.0 system which treats each instance in the training set as a rule, and then makes rules more general until classification accuracy starts to suffer.

Salzberg [1991] introduced the *Nested Generalized Exemplar* (NGE) theory, implemented in a system called *EACH*, in which hyperrectangles are used to take the place of one or more instances, thus reducing storage requirements. Wettschereck & Dietterich, [1995; Dietterich, 1994] introduced a hybrid nearest-neighbor and nearest-hyperrectangle algorithm that uses hyperrectangles to classify input vectors if they fall inside a hyperrectangle and kNN to classify inputs that are not covered by any hyperrectangle.

Dasarathy & Sheela [1979] introduced a composite classifier system consisting of a linear classifier similar to the *perceptron* [Rosenblatt, 1959] to quickly classify inputs that have a simple linear separation and a nearest neighbor rule to classify inputs that fall in the non-linearly separable portion of the input space. It uses less storage and is faster than the nearest neighbor algorithm but is more accurate than the perceptron.

A more in-depth survey of instance pruning techniques is presented in Chapter 7, where new reduction methods are also presented.

4.4. Weighting Techniques

One of the disadvantages of the basic nearest neighbor rule is its static nature. Once it has read in the training set, its learning is complete, and though it often generalizes well, it cannot perform adjustments to improve performance. This is in stark contrast to the operation of neural networks such as backpropagation networks that iterate and adjust weights to fit the problem.

There are several kinds of weights that can be used with nearest-neighbor models, including vote weighting and attribute weighting.

4.4.1. Vote Weighing

Vote weighting determines how much each neighbor is allowed to contribute to the decision as to what class should be assigned to an input vector. Dudani [1976] introduced a distance-weighted k-nearest neighbor rule in which nearer neighbors get more voting weight than further ones. The weight drops linearly from 1 at the distance of the first nearest neighbor to 0 at the distance of the kth neighbor. Keller, Gray & Givens [1985] developed a *Fuzzy k-NN* algorithm that also uses a distance-weighted voting scheme. However, instead of a linear drop-off, this algorithm uses an exponentially decreasing amount of weight as the distance increases. Also, it uses the voting weight for each class to provide an estimate of the confidence with which the classification is being made.

Radial Basis Function (RBF) networks [Broomhead & Lowe, 1988; Chen, 1991; Renals & Rohwer, 1989; Wong, 1991] also perform distance-weighted voting. Nodes in RBF networks output an activation that depends on the distance of an input vector to the node's center point. The activation typically drops off with distance according to a gaussian or exponential function. In probabilistic neural networks [Specht, 1992], the activation for each class is summed, which in essence allows each node to contribute some amount of voting weight for its desired output class.

Similar distance-weighted methods are used in Chapter 9, but several vote-weighting functions are provided, including those with linear, exponential and gaussian shapes. Confidence levels similar to those used in the *Fuzzy k-NN* method are also used, but they are combined with cross-validation accuracy estimates in such a way as to make them useful for tuning system parameters.

4.4.2. Attribute Weighting

Attribute weights can be used to avoid the detrimental effects of irrelevant and redundant input attributes. They can also be used to increase the influence of more important input attributes.

Wettschereck, Aha and Mohri [1995] provide an excellent survey of many attribute-weighting techniques. The most successful methods are the so-called *wrapper* models, which are those that use performance feedback (such as classification accuracy estimates) in making decisions on what weights to choose. These models include *incremental hill-climbing* methods such as *EACH* [Salzberg, 1991], which tunes attribute weights by a fixed amount after the presentation of each instance, and *IB4* [Aha, 1992], which finds a different set of attribute weights for each class.

Kelly & Davis [1991] used a *genetic algorithm* to find continuously valued feature weights, and Skalak [1994] used *random mutation hill-climbing* (a simplified genetic algorithm) to select binary feature weights.

Lowe [1995] used a distance-weighted *variable kernel similarity metric* (VSM) that used a conjugate gradient approach to minimize leave-one-out cross-validation classification error of the training set. The derivative of this error with respect to each attribute weight was used to guide the search for good attribute weights.

Two common methods for doing *attribute selection* (i.e., finding binary attribute weights) are *forward selection* and *backward elimination* [Miller, 1990]. Moore & Lee [1993] present a method using a *schemata* search that allows feature sets to be found even when there are correlations between attributes that can only be detected when certain attributes are combined.

It is possible to use more local weighting schemes. Aha & Goldstone [1990; 1992] developed a model called *Generalized Concept Model with Instance-Specific Weights* (GCM-ISW) that in addition to global attribute weights uses a different set of attribute weights for each instance. The weight used in computing distances is interpolated between the global and local weight for each instance depending on the distance for that attribute so that more local weights are used in parts of the input space nearby an instance. This has the effect of warping the input space and makes it possible for attributes to vary in their relevance throughout the input space.

This dissertation presents models that use a variety of vote-weighting schemes and also uses a real-valued schemata search to determine effective attribute weights that help to avoid the effects of irrelevant and redundant attributes.

5. Author's Previous Work

Some of the work presented in this dissertation is built upon a foundation of work done previously by the author. A paper entitled "The Importance of Using Multiple Styles of Generalization" [Wilson & Martinez, 1993a] empirically supported the theoretical work done by Schaffer [1993] and others showing that no single bias or learning algorithm can solve all problems well. It explored how having multiple algorithms to choose from may yield improved accuracy.

In addition, distance functions, discretization, voting schemes, weighting methods, inductive rules and confidence levels were explored in [Wilson & Martinez, 1993b] and in a Master's Thesis entitled *Prototype Styles of Generalization* [Wilson, 1994].

The work in this dissertation extends far beyond that presented in the thesis or earlier papers, but some of the background and intuition into instance-based learning algorithms comes from this previous work.

References

Aha, David W., (1992). "Generalizing from Case Studies: A Case Study," In *Proceedings of the Ninth International Conference on Machine Learning (MLC-92)*, Aberdeen, Scotland: Morgan Kaufmann, pp. 1-10.

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.

Aha, David W., and Robert L. Goldstone, (1990). "Learning Attribute Relevance in Context in Instance-Based Learning Algorithms," in *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA: Lawrence Erlbaum, pp. 141-148.

Aha, David W., and Robert L. Goldstone, (1992). "Concept Learning and Flexible Weighting," in *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, IN: Lawrence Erlbaum, pp. 534-539.

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

Batchelor, Bruce G., (1978). *Pattern Recognition: Ideas in Practice*. New York: Plenum Press, pp. 71-72.

Biberman, Yoram, (1994). A Context Similarity Measure. *In Proceedings of the European Conference on Machine Learning (ECML-94)*. Catalina, Italy: Springer Verlag, pp. 49-63.

Broomhead, D. S., and D. Lowe (1988). Multi-variable functional interpolation and adaptive networks. *Complex Systems*, vol. 2, pp. 321-355.

Buntine, Wray L., and Andreas S. Weigend, (1993). "Computing Second Derivatives in Feed-Forward Networks: a Review," *IEEE Transactions on Neural Networks*.

Cameron-Jones, R. M., (1995). Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing. *In Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence*, pp. 99-106.

Carpenter, Gail A., and Stephen Grossberg, (1987). "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115.

Chang, Chin-Liang, (1974). "Finding Prototypes for Nearest Neighbor Classifiers," *IEEE Transactions on Computers*, vol. 23, no. 11, November 1974, pp. 1179-1184.

Chen, S., C. R. N. Cowen, and P. M. Grant, (1991). "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 302-309.

Clark, Peter, and Tim Niblett, (1989). "The CN2 Induction Algorithm," *Machine Learning*, vol. 3, pp. 261-283.

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

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.

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

Dasarathy, Belur V., and Belur V. Sheela, (1979). "A Composite Classifier System Design: Concepts and Methodology," *Proceedings of the IEEE*, vol. 67, no. 5, May 1979, pp. 708-713.

Deng, Kan, and Andrew W. Moore, (1995). "Multiresolution Instance-Based Learning," to appear in *The Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95).*

Diday, Edwin, (1974). Recent Progress in Distance and Similarity Measures in Pattern Recognition. *Second International Joint Conference on Pattern Recognition*, pp. 534-539.

Dixon, John K., (1979). “Pattern Recognition with Partly Missing Data,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 10, October 1979, pp. 617-621.

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).*

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

Dudani, Sahibsingh A., (1976). “The Distance-Weighted *k*-Nearest-Neighbor Rule,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, no. 4, April 1976, pp. 325-327.

Fahlman, Scott E., (1988). “Faster-Learning Variations on Back-Propagation: An Empirical Study,” in D. S. Touretzky et al. (eds.), *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann.

Fahlman, Scott E., (1990). “The Cascade-Correlation Learning Architecture,” in D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann.

Gates, G. W. (1972). “The Reduced Nearest Neighbor Rule,” *IEEE Transactions on Information Theory*, vol. IT-18, no. 3, pp. 431-433.

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.

Hart, P. E., (1968). “The Condensed Nearest Neighbor Rule,” Institute of Electrical and Electronics Engineers Transactions on Information Theory, vol. 14, pp. 515-516.

Hecht-Nielsen, R., (1987). Counterpropagation Networks. *Applied Optics*, vol. 26, no. 23, pp. 4979-4984.

Keller, James M., Michael R. Gray, and James A. Givens, Jr., (1985). “A Fuzzy *K*-Nearest Neighbor Algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, no. 4, July/August 1985, pp. 580-585.

Kelly, J. D., Jr., and L. Davis, (1991). “A hybrid genetic algorithm for classification,” in *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia: Morgan Kaufmann, pp. 645-650.

Kibler, D., and David W. Aha, (1987). “Learning representative exemplars of concepts: An initial case study.” *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, CA: Morgan Kaufmann, pp. 24-30.

Kohonen, Teuvo, (1990). The Self-Organizing Map. *In Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464-1480.

Lebowitz, Michael, (1985). “Categorizing Numeric Information for Generalization,” *Cognitive Science*, vol. 9, pp. 285-308.

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

Lowe, David G., (1995). “Similarity Metric Learning for a Variable-Kernel Classifier,” *Neural Computation*., vol. 7, no. 1, pp. 72-85.

Macleod, James E. S., Andrew Luk, and D. Michael Titterington, (1987). “A Re-Examination of the Distance-Weighted *k*-Nearest Neighbor Classification Rule,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 17, no. 4, July/August 1987, pp. 689-696.

Martinez, Tony R., (1986). “Adaptive Self-Organizing Logic Networks,” Ph.D. Dissertation, University of California Los Angeles Technical Report - CSD 860093, (274 pp.).

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

Martinez, Tony R., and Douglas M. Campbell, (1991a). “A Self-Adjusting Dynamic Logic Module,” *Journal of Parallel and Distributed Computing*, vol. 11, no. 4, pp. 303-313.

Martinez, Tony R., and Douglas M. Campbell, (1991b). “A Self-Organizing Binary Decision Tree for Incrementally Defined Rule Based Systems,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 5, pp. 1231-1238, 1991.

Martinez, Tony R., and J. J. Vidal, (1988). “Adaptive Parallel Logic Networks,” *Journal of Parallel and Distributed Computing*, vol. 5, no. 1, pp. 26-58.

Martinez, Tony R., Douglas M. Campbell, Brent W. Hughes, (1994). “Priority ASOCS,” *Journal of Artificial Neural Networks*, vol. 1, no. 3, pp. 403-429.

Martinez, Tony, (1989). “Neural Network Applicability: Classifying the Problem Space,” *Proceedings of the IASTED International Symposium on Expert Systems and Neural Networks*, pp. 12-15.

Møller, Martin F., (1993). “A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning,” *Neural Networks*, vol. 6, pp. 525-533.

Michalski, Ryszard S., (1969). “On the quasi-minimal solution of the general covering problem,” *Proceedings of the Fifth International Symposium on Information Processing*, Bled, Yugoslavia, pp. 12-128.

Michalski, Ryszard S., Robert E. Stepp, and Edwin Diday, (1981). A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts. *Progress in Pattern Recognition*, vol. 1, Laveen N. Kanal and Azriel Rosenfeld (Eds.). New York: North-Holland, pp. 33-56.

Miller, A. J., (1990). *Subset selection in Regression*, Chapman and Hall, 1990.

Mitchell, Tom M., (1980). “The Need for Biases in Learning Generalizations,” in J. W. Shavlik & T. G. Dietterich (Eds.), *Readings in Machine Learning*, San Mateo, CA: Morgan Kaufmann, 1990, pp. 184-191.

Mitchell, Tom M., (1997). *Machine Learning*, McGraw Hill.

Mohri, Takao, and Hidehiko Tanaka, “An Optimal Weighting Criterion of Case Indexing for Both Numeric and Symbolic Attributes. In D. W. Aha (Ed.), Case-Based Reasoning: Papers from the 1994 Workshop, Technical Report WS-94-01. Menlo Park, CA: AIII Press, pp. 123-127.

Moore, Andrew W., and Mary S. Lee, (1993). “Efficient Algorithms for Minimizing Cross Validation Error,” In *Machine Learning: Proceedings of the Eleventh International Conference*, Morgan Kaufmann.

Nadler, Morton, and Eric P. Smith, (1993). *Pattern Recognition Engineering*. New York: Wiley, pp. 293-294.

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.

Quinlan, J. R., (1986). “Induction of Decision Trees”, *Machine Learning*, vol. 1, pp. 81-106.

Quinlan, J. R., (1989). “Unknown Attribute Values in Induction,” *Proceedings of the 6th International Workshop on Machine Learning*, San Mateo, CA: Morgan Kaufmann, pp. 164-168.

Quinlan, J. R., (1993). *C4.5: Programs for Machine Learning*, San Mateo, CA: Morgan Kaufmann.

Rachlin, John, Simon Kasif, Steven Salzberg, David W. Aha, (1994). “Towards a Better Understanding of Memory-Based and Bayesian Classifiers,” in *Proceedings of the Eleventh International Machine Learning Conference,* New Brunswick, NJ: Morgan Kaufmann, pp. 242-250.

Renals, Steve, and Richard Rohwer, (1989). “Phoneme Classification Experiments Using Radial Basis Functions,” *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN’89)*, vol. 1, pp. 461-467.

Ritter, G. L., H. B. Woodruff, S. R. Lowry, and T. L. Isenhour, (1975). "An Algorithm for a Selective Nearest Neighbor Decision Rule," *IEEE Transactions on Information Theory*, vol. 21, no. 6, November 1975, pp. 665-669.

Rosenblatt, Frank, (1959). *Principles of Neurodynamics*, New York, Spartan Books.

Rudolph, George L., and Tony R. Martinez, "An Efficient Static Topology For Modeling ASOCS," *International Conference on Artificial Neural Networks*, Helsinki, Finland. In *Artificial Neural Networks*, Kohonen, et. al. (Eds.), Elsevier Science Publishers, North Holland, pp. 729-734, 1991.

Rumelhart, D. E., and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, 1986.

Salzberg, Steven, (1991). "A Nearest Hyperrectangle Learning Method," *Machine Learning*, vol. 6, pp. 277-309.

Salzberg, Steven, Arthur Delcher, David Heath, and Simon Kasif, (1995). "Best-Case Results for Nearest Neighbor Learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 6, June 1995, pp. 599-608.

Schaffer, Cullen, (1993). "Selecting a Classification Method by Cross-Validation," *Machine Learning*, vol. 13, no. 1.

Schaffer, Cullen, (1994). "A Conservation Law for Generalization Performance," In *Proceedings of the Eleventh International Conference on Machine Learning (ML'94)*, Morgan Kaufmann, 1994.

Skalak, David B., (1994). "Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms," *Proceedings of the Eleventh International Conference on Machine Learning (ML'94)*.

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.

Specht, Donald F., (1992). "Enhancements to Probabilistic Neural Networks," in *Proceedings International Joint Conference on Neural Networks (IJCNN '92)*, vol. 1, pp. 761-768.

Sproull, Robert F., (1991). "Refinements to Nearest-Neighbor Searching in *k*-Dimensional Trees," *Algorithmica*, vol. 6, pp. 579-589.

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

Swonger, C. W., (1972). "Sample Set Condensation for a Condensed Nearest Neighbor Decision Rule for Patter Recognition," *Frontiers of Pattern Recognition*, edited by S. Watanabe, Academic Press, pp. 511-519.

Ting, Kai Ming, (1994). "Discretization of Continuous-Valued Attributes and Instance-Based Learning," Technical Report no. 491, Basser Department of Computer Science, The University of Sydney, Australia.

Ting, Kai Ming, (1996). "Discretisation in Lazy Learning Algorithms," to appear in the special issue on Lazy Learning in *Artificial Intelligence Review*.

Tomek, Ivan, (1976). "An Experiment with the Edited Nearest-Neighbor Rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 6, June 1976, pp. 448-452.

Turney, Peter, (1994). Theoretical Analyses of Cross-Validation Error and Voting in Instance-Based Learning. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, pp. 331-360.

Ventura, Dan, (1995). *On Discretization as a Preprocessing Step for Supervised Learning Models*, Master's Thesis, Brigham Young University, 1995.

Ventura, Dan, and Tony R. Martinez (1995). "An Empirical Comparison of Discretization Methods." *In Proceedings of the Tenth International Symposium on Computer and Information Sciences*, pp. 443-450.

Wasserman, Philip D., (1993). *Advanced Methods in Neural Computing*, New York, NY: Van Nostrand Reinhold, pp. 147-176.

Wess, Stefan, Klaus-Dieter Althoff and Guido Derwand, (1994). "Using *k*-d Trees to Improve the Retrieval Step in Case-Based Reasoning," Stefan Wess, Klaus-Dieter Althoff, & M. M. Richter (Eds.), *Topics in Case-Based Reasoning*. Berlin: Springer-Verlag, pp. 167-181.

Wettschereck, Dietrich, (1994). "A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm", *To appear in the Proceedings of the 7th European Conference on Machine Learning*.

Wettschereck, Dietrich, and Thomas G. Dietterich, (1995). "An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms," *Machine Learning*, vol. 19, no. 1, pp. 5-28.

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.

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.

Widrow, Bernard, Rodney Winter, "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition," *IEEE Computer Magazine*, pp. 25-39, March 1988.

Wilson, D. Randall, and Tony R. Martinez, (1993a). "The Importance of Using Multiple Styles of Generalization," *Proceedings of the First New Zealand International Conference on Artificial Neural Networks and Expert Systems (ANNES)* , pp. 54-57.

Wilson, D. Randall, and Tony R. Martinez, (1993b). "The Potential of Prototype Styles of Generalization," *The Sixth Australian Joint Conference on Artificial Intelligence (AI '93)*, pp. 356-361.

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

Wilson, Dennis L., (1972). "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, July 1972, pp. 408-421.

Wolpert, David H., (1993). "On Overfitting Avoidance as Bias," Technical Report SFI TR 92-03-5001, Santa Fe, NM: The Santa Fe Institute. Internet: ftp://ftp.santafe.edu/pub/dhw_ftp/overfitting.avoidance.ps.Z.

Wong, Yiu-fai, (1991). "How Gaussian Radial Basis Functions Work," *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'91)*, vol. 2, pp. 133-138.

Zarndt, Frederick, (1995). *A Comprehensive Case Study: An Examination of Connectionist and Machine Learning Algorithms*, Master's Thesis, Brigham Young University.

Zhang, Jianping, (1992). "Selecting Typical Instances in Instance-Based Learning," *Proceedings of the Ninth International Conference on Machine Learning*.

Chapter 3

Bias and the Probability of Generalization

"In theory, there is no difference between theory and practice,
but in practice, there is."

Submitted to
International Conference on Intelligent Information Systems (IIS'97), 1997.

Abstract

In order to be useful, a learning algorithm must be able to generalize well when faced with inputs not previously presented to the system. A bias is necessary for any generalization, and as shown by several researchers in recent years, no bias can lead to strictly better generalization than any other when summed over all possible functions or applications. This paper provides examples to illustrate this fact, but also explains how a bias or learning algorithm can be "better" than another in practice when the probability of the occurrence of functions is taken into account. It shows how domain knowledge and an understanding of the conditions under which each learning algorithm performs well can be used to increase the probability of accurate generalization, and identifies several of the conditions that should be considered when attempting to select an appropriate bias for a particular problem.

1. Introduction

An inductive learning algorithm learns from a collection of examples, and then must try to decide what the output of the system should be when a new input is received that was not seen before. This ability is called *generalization*, and without this ability, a learning algorithm would be of no more use than a simple look-up table.

For the purpose of this paper, we assume that we have a *training set*, T, consisting of *n instances*. Each instance has an input vector consisting of one value for each of *m* input attributes, and an *output value*. The output value can be a continuous value, in the case of regression, or a discrete class, in the case of classification. Most of the examples in this paper will use classification for simplicity, but the discussion applies to regression as well.

In order to generalize, an algorithm must have a *bias*, which Mitchell [1980] defined as "a rule or method that causes an algorithm to choose one generalized output over another." Without a bias, an algorithm can only provide a correct output value in response to an input vector it has seen during learning (and even that assumes a consistent, correct training set). For other input vectors it would simply have to admit that it does not know what the output value should be. A bias is therefore crucial to a learning algorithm's ability to generalize.

However, selecting a good bias is not trivial, and in fact may be considered to be one of the main areas of research in the fields of machine learning, neural networks, artificial intelligence, and other related fields.

Biases are usually not explicitly defined, but are typically inherent in a learning algorithm that has some intuitive and/or theoretical basis for leading us to believe it will be successful in providing accurate generalization in certain situations. It is in fact difficult to give a precise definition of the bias of even a well-understood learning model, except in terms of how the algorithm itself works. Parameters of an algorithm also affect the bias.

Dietterich [1989], Wolpert [1993], Schaffer [1994], and others have shown that no bias can achieve higher generalization accuracy than any other bias when summed over all possible applications. This seems somewhat disconcerting, as it casts an apparent blanket of hopelessness over research focused on discovering new, "better" learning algorithms. Section 2 presents arguments and examples illustrating this *Conservation Law for Generalization Performance* [Schaffer, 1993]. Section 3 discusses the *bias of simplicity*, and illustrates how one bias *can* lead to better generalization than another, both theoretically and empirically, when functions are weighted according to their probability of occurrence. This probability is related to how much regularity a function contains and whether it is an important kind of regularity that occurs often in real world problems.

Table 1. Truth table for 2-input 1-output boolean functions.

X:	0 0 1 1
Y:	0 1 0 1
ZERO	0 0 0 0
AND	0 0 0 1
X ^ ~Y	0 0 1 0
X	0 0 1 1
Y ^ ~X	0 1 0 0
Y	0 1 0 1
XOR	0 1 1 0
OR	0 1 1 1
NOR	1 0 0 0
EQUAL	1 0 0 1
~Y	1 0 1 0
X | ~Y	1 0 1 1
~X	1 1 0 0
~X | Y	1 1 0 1
NAND	1 1 1 0
ONE	1 1 1 1

The key to improved generalization is to first have a powerful collection of biases available that generalize well on many problems of interest, and then to use any knowledge of a particular application domain we may have to choose a bias (i.e., a learning algorithm and its parameters) that is appropriate for that domain. Section 4 gives a list of conditions to consider when trying to match an application domain with a learning algorithm. It is important to understand how various learning algorithms behave under these conditions [Aha, 1992b] so that applications can be matched with an appropriate bias. Section 5 draws conclusions from the discussion.

2. Why One Bias Cannot be "Better" than Another

The Conservation Law of Generalization [Schaffer, 1994] states that no bias (or learning algorithm) can achieve higher generalization than any other when summed over all possible learning problems.

In order to illustrate this law, consider all of the possible 2-input 1-output binary problems, listed in Table 1.

Table 2. Example of a training set.

0	0	->	0
0	1	->	1
1	0	->	1
1	1	->	?

A name for each function is given next to its output for each input pattern. We refer to functions either by their name (e.g., "OR"), or their truth values (e.g., "0111"). Suppose that for a particular problem our training set contains three out of the four possible input patterns as shown in Table 2.

The last entry (1,1->?) is the only pattern that is not preclassified in this example. We can think of the training set as a template ("011?") used to see which functions are consistent with it, where a question mark indicates that the output is unknown for the corresponding input pattern. Of the 16 possible 2-input binary functions, two are consistent with the supplied training set, namely, XOR ("0110") and OR ("0111"). Unfortunately, there is no way to tell from the data which of these two functions is correct in this case.

Consider three simple biases that could be applied here:

1. *Most Common (MC)*: select the most common output class.
2. *Least Common (LC)*: select the least common output class.
3. *Random*: randomly choose an output class.

The first (MC) would choose an output of 1, and thus choose the "OR" function. The second (LC) would choose an output of 0, and thus choose the "XOR" function. The Random function could choose either one.

If the function is really "OR", then MC would be correct, and LC would be wrong. If the function is really "XOR", then the opposite would be true. The average generalization accuracy for MC, LC and Random over the two possible functions is the same: 50%. Regardless of which output a bias chooses given the three known input patterns, if it is correct for one function, it must be wrong on the other.

Cross-validation [Schaffer, 1993] is often used to help select among possible biases, e.g., to select which parameters an algorithm should use (which affects the bias), or to select which algorithm to use for a problem. However, cross-validation is still a bias, and thus cannot achieve better-than-random generalization when summed over all functions.

As an example, suppose we hold out the first training pattern for evaluating our available biases, similar to what is done in cross-validation. The second and third input patterns yield a template of "?11?", which is matched by four functions: "0110", "1110", "0111", and "1111". MC would choose the function "1111" as its estimated function, while LC would choose the function "0110". In this case, LC looks like the better bias, since it generalized from the subset to the training set more correctly than MC did.

If the true underlying function is "0110", then cross-validation will have chosen the correct bias. However, the fact remains that if the true function is "0111", MC rather than LC would be the correct choice of bias to use. Again, the average generalization accuracy over the possible functions is 50%.

This example also illustrates that though it might be tempting to think so, even the addition of "fresh data" to our original training set does not help in determining which learning algorithm will generalize more accurately on unseen data for a particular application. If this were not true, then part of the original training set could be held out and called "fresh." Of course, when more data is available a larger percentage of input vectors can be memorized and thus guaranteed to be correct (assuming consistent, correct training data), but this still does not help generalization on unseen input patterns when considering all the possible functions that are consistent with the observed data.

Given an m-input 1-output binary problem, there are 2^m possible input patterns and 2^{2^m} possible functions to describe the mapping from input vector to output class. Given n training instances, there will be $(2^{2^m}/2^n)$=2^(2^m-n) possible functions to describe the mapping. For example, a 10-input binary training set with 1000 (of the possible 1024) unique training instances specified would still be consistent with 2^(1024-1000)=2^{24}≈4 million different functions, even though almost all of the possible instances were specified. Every training instance added to the training set cuts the number of possible functions by half, but this also implies that every possible input pattern *not* in the training set doubles the number of possible functions consistent with the data.

For functions with more than two possible values for each input variable and output value, the problem becomes worse, and when continuous values are involved, there are an infinite number of functions to choose from.

Some bias is needed to choose which of all possible functions to use, and some guidance is needed to decide which bias to use for a particular problem, since no bias can be better than any other for all possible problems.

3. Why One Bias *Can* be "Better" than Another

Section 2 illustrated how every bias has the same average generalization accuracy, regardless of which function it chooses to explain a set of training instances. This seems somewhat disconcerting, as it casts an apparent blanket of hopelessness over research focused on discovering new, "better" learning algorithms. This section explains why some algorithms can have higher generalization accuracy than others in practice.

Let F be the set of functions consistent with a training set, and $|F|$ be the number of such functions in F. Then the *theoretical average accuracy* (i.e., that discussed in Section 2) is given by:

$$ta(b) = \frac{\sum_{f \in F} g(b,f)}{|F|} = C \tag{1}$$

where $g(b,f)$ is the average generalization accuracy of a bias b on a function f, and C is some constant (0.5 for functions with boolean outputs), indicating that the theoretical average accuracy is the same for all biases.

3.1. Functions are Not Equally Important

The theoretical average accuracy treats all functions as equally important. In practice, however, some functions are much more important that others. Functions have different amounts of regularity, and also have different kinds of regularity (as discussed in more detail in following sections). Only a vanishingly small fraction of all functions have large amounts of regularity, and yet most of the problems we are interested in solving have strong regularities in their underlying functions (assuming a good set of input attributes is used to describe the problem). Such functions are therefore much more important in practice than the remaining ones.

If one bias achieves higher average generalization accuracy than another on these important functions, then it is "better" (in practice) than the other, even though it must do correspondingly worse on some problems that are unimportant to us.

The importance of a function is related closely to its likelihood [Wolpert, 1993] of occurring in practice. If a particular kind of regularity occurs often in problems of interest in the real world, then functions that contain this kind of regularity will have a higher probability of occurring in practice than others.

If the generalization accuracy of each function is weighted by the probability of its occurrence (and thus indirectly by its importance), the *practical average accuracy* is given as:

$$pa(b) = \frac{\sum_{f \in F} p(f) g(b,f)}{\sum_{f \in F} p(f)} \tag{2}$$

Where $p(f)$ is the probability of each function f in F occurring in practice. Using this measure, a bias that generalizes well on common functions and poorly on uncommon functions will have a higher practical average accuracy than a bias with the opposite generalization behavior, even though both have the same theoretical average accuracy.

Since there are an infinite number of functions, the above functions are not computed explicitly, but they do help to explain why many learning algorithms—such as C4.5 [Quinlan, 1993], k-nearest neighbor [Cover & Hart, 1967], and backpropagation neural networks [Rumelhart & McClelland, 1986]—have empirically been able to generalize much better than random on applications of interest. These and other learning models have a bias that is appropriate for many applications that occur in practice, and thus achieve good accuracy in such situations.

3.2. Bias of simplicity

One bias that is in wide use in a variety of learning algorithms is the *bias of simplicity* (*Occam's Razor*). When there are multiple possible explanations for the same data, this bias tends to choose the simplest one. The bias of simplicity has been employed in many different learning algorithms, and with good success. We as humans use the bias of simplicity quite often in trying to make sense of complex data.

The success of the bias of simplicity suggests that many of the problems that we try to solve with learning algorithms have underlying regularities that these algorithms are able to discover. Put another way, the probability of simple functions is higher than that of more complex functions, so a bias that favors simplicity will have a higher probability of generalizing correctly than one that does not.

One problem with the bias of simplicity is that there is no fixed definition of what is "simple." For example, the XOR problem can be described in English quite simply, i.e., "odd number of 1's in the input vector." However, describing this in a logic equation is much more complex (when the number of inputs is large) than some functions which would be more lengthy to describe in English.

Often the representation language can have a great impact on the simplicity of a concept description. In fact, this is one way in which learning algorithms differ. Many algorithms seek to choose the simplest concept description that is (approximately) consistent with the training set, but do so according to their own representational language, which in turn influences what bias is used in choosing a concept description.

Thus, even when using the bias of simplicity, one must decide upon an algorithm that will choose the simplest concept description in its own representational language that will provide higher generalization accuracy than the other algorithms. Put another way, each algorithm is good at identifying only certain *kinds* of regularity.

If we have an *a priori* knowledge of the problem's underlying function, we may be able to decide which algorithm is most likely to be suitable for it. Of course, knowing the underlying function makes the learning algorithm unnecessary in most cases. More likely, we will have some intuition as to what the problem's underlying function *probably* looks like, or what it *approximately* looks like, in which case we can choose an algorithm that appears to be well suited for such a problem.

3.3. Additional Information

Schaffer [1994] mentioned that the only way to choose one algorithm over another for a particular problem and expect it to generalize more accurately is if we have additional information about the problem besides the raw training data. This additional information cannot be in the form of additional training instances, for these tell us only what the output should be at the additional specific points. Rather, the additional information should be general knowledge, intuition, or even reasonable assumptions regarding the underlying problem and the mapping from its inputs to outputs.

For example, knowing whether the input values are linear or nominal may be important. Knowing that "values nearer to each other are more likely to correspond to similar output classes than values far from each other" would indicate a geometrically based problem that a variety of learning algorithms are well suited for. Knowing that the problem is somewhat similar to one that was solved successfully by a particular learning algorithm might be helpful.

Such intuition or knowledge does more than just specify what the output should be at additional points in the input space. Rather, it gives a hint (i.e., an indication or bias) of how the function behaves across the entire input space, thus providing information and guidance in areas of the input space that are not explicitly mapped to output values.

In essence, such knowledge increases the probability that a learning algorithm will be applied to a problem that it is appropriate for, and thus raises the average practical generalization accuracy of that algorithm.

Thus general knowledge about a problem *can* be used to select an appropriate bias, and has the potential to improve generalization accuracy, even if a strict examination of the data cannot.

To see how the practical average accuracy is affected by the use of additional knowledge, consider a meta-bias M that works as follows.

> Learn as much as possible about a problem domain, and use this knowledge to select a bias b_i (from a set B of available biases) that appears to be most appropriate for the problem.

The average accuracy of the bias M for a given function f is given as:

$$g(M,f) = \sum_{i=1}^{|B|} p(b_i|K,f)g(b_i,f) \tag{3}$$

where K is the domain knowledge and knowledge of the characteristics of the biases in B; $p(b_i|K,f)$ is the probability (averaged over all possible training sets for f) of choosing bias b_i given an underlying function f and our knowledge K; $|B|$ is the number of available biases; and $g(b_i,f)$ is the average accuracy for a particular bias b_i for the function f. The set B is limited in a practical setting by what biases are available to those who are trying to solve real problems (i.e., what algorithms they are aware of and can implement and/or use).

The average accuracy $g(b_i,f)$ for each bias is fixed for a given function f, but the probability $p(b_i|K,f)$ of choosing the bias depends on our understanding of a particular application domain and of the various available biases.

Thus, there are two ways to increase practical generalization accuracy using this meta-bias M. The first is to find additional algorithms and/or parameters to add to B that yield high values of $g(b_i,f)$ for important classes of functions, especially those not handled well by other algorithms already in B, and to learn enough about the new biases to apply them appropriately. This can be done by introducing new learning algorithms and modifying existing algorithms to achieve higher generalization accuracy on at least a subset of real-world learning problems, and by identifying characteristics of problems for which the new bias is successful.

The second way to increase practical generalization accuracy using this meta-bias is to increase our understanding of the capabilities of each bias and increase our ability to identify characteristics of applications. This allows us to increase the probability $p(b_i|K,f)$ of selecting biases that are likely to achieve high generalization accuracy $g(b_i,f)$ while decreasing the probability of choosing inappropriate biases that would result in lower accuracy.

It is therefore very important to know under what conditions each algorithm generalizes well, and how to determine whether a particular problem exhibits such conditions. Section 4 gives a list of conditions to consider when trying to match an application domain with a learning algorithm.

4. Characteristics of Algorithms and Applications

Each learning algorithm has certain conditions under which it performs well. For example, nearest neighbor classifiers work well when similar input vectors correlate well with similar output classes. Using the two-input binary example from Section 2, given a template "011?", a nearest neighbor classifier would assume this is the "OR" function "0111", since the unspecified pattern, "11" is closer to "01" and "10" than to "00".

The XOR function ("0110"), on the other hand, violates the similarity criterion, because values nearer each other are actually more likely to be of *different* classes. Thus the nearest neighbor and other geometrically based algorithms are not appropriate for the XOR function, because they will provide random or worse generalization.

One way that models can be "improved" is by identifying conditions under which it does not perform well (e.g., by finding kinds of regularity that the algorithm cannot identify or represent), and then add the capability to handle such conditions when it is likely that they exist in an application. For example, the nearest neighbor algorithm is extremely sensitive to irrelevant attributes, so an extension to the basic algorithm that finds attribute weights or removes irrelevant attributes would be likely to improve generalization when there is a strong likelihood that there are irrelevant attributes in the problem. Again, our knowledge of the application domain, the source of the data, and other such knowledge can help to identify when such conditions are likely.

When this kind of knowledge is available about an application, we can match this information against our knowledge of various learning algorithms (or the effect of various parameters in an algorithm) to choose one that is *appropriate*, i.e., one designed to handle the aspects we know about the problem, and thus one likely to generalize well on it.

4.1. Characteristics of Applications

This section presents a list of issues that can be used to decide whether an algorithm is appropriate for an application. One useful area of research in machine learning is to identify how each learning algorithm addresses such issues, as well as how to identify characteristics of applications in relation to each issue.

The following list is not exhaustive, but is meant to serve as a starting point in identifying characteristics of an application.

4.1.1. Number of input attributes (dimensionality).

Some applications have a large number of input attributes, which can be troublesome for algorithms that suffer from the "curse of dimensionality." For example, *k*-d trees [Wess, Althoff & Derwand, 1994] for speeding up searches in nearest neighbor classifiers are not effective when the dimensionality grows too large [Sproull, 1991]. On the other hand, some algorithms can make use of the additional information to improve generalization, especially if they have a way of ignoring attributes that are not useful.

4.1.2. Type of input attributes.

Input attributes can be nominal (discrete, unordered), linear (discrete, but ordered), or continuous (real-valued), and applications can have input attributes that are all of one type or a mixture of different kinds of attributes [Wilson & Martinez, 1997]. Some models are designed only to handle one kind of attribute. For example, some models cannot handle continuous attributes and must therefore *discretize* [Lebowitz, 1985; Schlimmer, 1987] such attributes before using them.

4.1.3. Type of output.

Output values can be continuous or discrete. Many learning models are designed only for classification, and thus can handle only discrete outputs, while others perform regression and are appropriate for continuous outputs.

4.1.4. Noise.

Errors can occur in input values or output values, and can result from measurement errors, corruption of the data, or unusual occurrences that are correctly recorded. Noise-reduction algorithms such as pruning techniques in decision trees [C4.5] or the use of $k>1$ in the *k*-nearest neighbor algorithm [Cover & Hart, 1967] can help reduce the effect of noisy instances, though such techniques can also hurt generalization in some cases, especially when noise is not present.

Many applications also have missing values (or "don't know" values) that must be dealt with in a reasonable manner [Quinlan, 1989].

4.1.5. Irrelevant attributes.

Some learning models such as C4.5 [Quinlan, 1993] are quite good at ignoring irrelevant attributes, while others, such as the basic nearest-neighbor algorithm, are extremely sensitive to them, and require modifications [Aha, 1992a; Wilson & Martinez, 1996] to handle them appropriately.

4.1.6. Shape of decision surface.

The shape of the decision surface in the input space can have a major effect on whether an algorithm can solve the problem efficiently. Many models are limited in the kinds of decision surfaces they can generate. For example, decision trees and rule-based systems often have axis-aligned hyperrectangular boundaries; nearest-neighbor classifiers can form decision boundaries made of the intersection of hyperplanes (each of which bisects the line between two instances of different classes); backpropagation neural networks form curved surfaces formed by an intersection of sigmoidal hypersurfaces [Lippmann, 1987].

Many problems have geometric decision surfaces such that points close together are grouped into the same class or have similar output values, and most learning algorithms do better with such problems. Others, like the XOR problem, do not have geometrically simple decision surfaces, and are thus difficult for many learning algorithms, though other representations like logic statements can sometimes be used to generalize in such cases.

Some problems also have overlapping concepts, which makes a rigid decision surface inappropriate. Models such as backpropagation networks that have a confidence associated with their decisions can be useful in such cases.

4.1.7. Data density.

The density of data can be thought of as either the proportion of possible input patterns that are included in the training set, or as the amount of training data available compared to the complexity of the decision surface.

4.1.8. Order of attribute-class correlations.

Some problems can be solved using low-order combinations of input attributes (e.g., the *Iris* database [Merz & Murphy, 1996] can be largely solved using only one of the inputs), while other problems can only be solved using combinations of several or all of the input attributes. Similarly, some models can do only linearly separable problems (e.g., perceptron [Widrow & Lehr, 1990]), though most do handle higher-order combinations of input values.

The first three criteria are usually easy to identify (number and types of input and output attributes). We also often have a feel for how accurate the data is that we have collected, and whether it is likely to contain some noise. Missing values are also easily identified.

Irrelevant attributes are usually difficult to identify, because sometimes an attribute will only correlate well with the output when combined in some higher-order way with other attributes. Such combinations are difficult to identify, since there are an exponential number of them to check for, and typically insufficient data to support strong conclusions about which combinations of attribute values are significant.

4.2. Characteristics of Learning Algorithms

Each of the issues listed in Section 4.1 identifies characteristics of applications to keep in mind when choosing a learning algorithm. It is certainly not trivial to obtain such information about applications, but hopefully at least some of the above information can be obtained about a particular application before deciding upon a learning algorithm to use on it.

In order for such information to be useful in choosing a learning algorithm, knowledge about individual learning models must also be available. One way to determine the conditions under which an algorithm will perform well is to use artificial data. Artificial data can be designed to test specific conditions such as noise-tolerance, non-axis-aligned decision boundaries, and so forth. Since the researcher has complete control over how such data is constructed, and knows what the underlying function really is, it can be modified in ways to test specific abilities.

However, it is still a necessity to test algorithms on real data, too, in order to see *how well* the algorithm works in typical real-world conditions. In addition, real-world data can be modified to see how changing certain conditions affects generalization ability or other aspects of the algorithm's performance. For example, to test noise tolerance, a real-world dataset can have noise added to it by randomly changing input or output values to see how fast generalization accuracy drops with an increasing level of noise. Similarly, irrelevant attributes can be added to see how a model handles them.

In addition to such empirical studies, theoretical conclusions can often be drawn from an examination of the learning algorithm itself. For example, the possible shapes of decision surfaces can often be derived from looking at an algorithm and the representation it uses for concept descriptions. Once the theoretical limits on the shape of the decision surface is determined, artificial functions can be used to see how well different surfaces can be approximated by a learning model. Some simple shapes that can be used as starting points include axis-aligned hyperrectangles, diagonal hyperplanes, and hyperspheres.

By using a combination of theoretical analysis, artificial data, real-world data, and artificially modified real-world data, much can be learned about each learning algorithm and the conditions under which it will fail or generalize well. When combined with knowledge about a particular application (outside of the raw training data), the probability of achieving high generalization can be substantially increased.

5. Conclusions

A learning algorithm needs a bias in order to generalize. No bias can achieve higher *theoretical* average generalization accuracy than any other when summed over all applications. However, some biases *can* achieve higher *practical* average generalization accuracy than others when their bias is more appropriate for those functions that are more likely to occur in practice, even if their bias is worse for functions that are less likely to occur.

In order to increase the probability of achieving high generalization accuracy, it is important to know what characteristics each learning algorithm has, and how an algorithm's parameters affect these characteristics, so that an appropriate algorithm can be chosen to handle each application. By increasing the probability that an appropriate bias will be chosen for each problem, the average practical generalization accuracy can be increased.

Research in machine learning and related areas should seek to identify characteristics of learning models, identify conditions for which each model is appropriate, and address areas of weakness among them. It should also continue to introduce new learning algorithms, improve existing algorithms, and indicate when such algorithms and improvements are appropriate. Research should also continue to explore ways of using knowledge outside of the raw training data to help decide what bias would be best for a particular application. By so doing, the chance for increased generalization accuracy in real-world situations can continue to be improved.

References

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

Aha, David W., (1992b). "Generalizing from Case Studies: A Case Study." In *Proceedings of the Ninth International Conference on Machine Learning (MLC-92)*, Aberdeen, Scotland: Morgan Kaufmann, pp. 1-10.

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, pp. 21-27.

Dietterich, Thomas G., (1989). "Limitations on Inductive Learning." In *Proceedings of the Sixth International Conference on Machine Learning*.

Lebowitz, Michael, (1985). "Categorizing Numeric Information for Generalization." *Cognitive Science*, vol. 9, pp. 285-308.

Lippmann, Richard P., (1987). "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, vol. 3, no. 4, pp. 4-22.

Merz, C. J., and P. M. Murphy, (1996). *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: http://www.ics.uci.edu/~mlearn/MLRepository.html.

Mitchell, Tom M., (1980). "The Need for Biases in Learning Generalizations." in J. W. Shavlik & T. G. Dietterich (Eds.), *Readings in Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1990, pp. 184-191.

Quinlan, J. R., (1989). "Unknown Attribute Values in Induction." In *Proceedings of the 6th International Workshop on Machine Learning*. San Mateo, CA: Morgan Kaufmann, pp. 164-168.

Quinlan, J. R., (1993). *C4.5: Programs for Machine Learning*, San Mateo, CA: Morgan Kaufmann.

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

Schaffer, Cullen, (1993). "Selecting a Classification Method by Cross-Validation." *Machine Learning*, vol. 13, no. 1.

Schaffer, Cullen, (1994). "A Conservation Law for Generalization Performance." In *Proceedings of the Eleventh International Conference on Machine Learning (ML'94)*, Morgan Kaufmann, 1994.

Schlimmer, Jeffrey C., (1987). "Learning and Representation Change." In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87)*, vol. 2, pp. 511-535.

Sproull, Robert F., (1991). "Refinements to Nearest-Neighbor Searching in k-Dimensional Trees." *Algorithmica*, vol. 6, pp. 579-589.

Wess, Stefan, Klaus-Dieter Althoff and Guido Derwand, (1994). "Using k-d Trees to Improve the Retrieval Step in Case-Based Reasoning." Stefan Wess, Klaus-Dieter Althoff, & M. M. Richter (Eds.), *Topics in Case-Based Reasoning*. Berlin: Springer-Verlag, pp. 167-181.

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

Wilson, D. Randall, and Tony R. Martinez, (1997). "Improved Heterogeneous Distance Metrics," *Journal of Artificial Intelligence Research*, vol. 6, no. 1, pp. 1-34.

Wilson, D. Randall, and Tony R. Martinez, "Instance-Based Learning with Genetically Derived Attribute Weights," *International Conference on Artificial Intelligence, Expert Systems and Neural Networks (AIE'96)*, pp. 11-14, August 1996.

Wolpert, David H., (1993). "On Overfitting Avoidance as Bias." Technical Report SFI TR 92-03-5001. Santa Fe, NM: The Santa Fe Institute.

Part II
Heterogeneous Distance Functions

"And who is my neighbor?"
—Luke 10:29.

Many applications have both linear and nominal attributes but previous distance functions did not handle both kinds of attributes satisfactorily. Chapter 4 introduces the *Heterogeneous Value Difference Metric* (HVDM), which uses Euclidean distance for linear attributes and the *Value Difference Metric* (VDM) for nominal attributes. This chapter then uses the HVDM in a radial basis function neural network.

Chapter 4 was published under the following reference.

Wilson, D. Randall, and Tony R. Martinez, "Heterogeneous Radial Basis Function Networks," *Proceedings of the International Conference on Neural Networks (ICNN'96)*, vol. 2, pp. 1263-1267, June 1996.

Chapter 5 introduces two additional heterogeneous distance functions that allow the VDM to be applied directly to continuously valued attributes, thus avoiding problems associated with normalization between nominal and continuous attributes. These two distance functions, the *Interpolated VDM* (IVDM) and the *Windowed VDM* (WVDM), were first published in the following conference paper.

Wilson, D. Randall, and Tony R. Martinez, (1996). "Value Difference Metrics for Continuously Valued Attributes," *International Conference on Artificial Intelligence, Expert Systems and Neural Networks (AIE'96),* pp. 74-78.

Chapter 5 discusses modifications to the normalization used by HVDM, presents the IVDM and WVDM distance functions, and provides in-depth comparisons between these new heterogeneous distance functions and several popular alternatives used previously. Chapter 5 was published in the *Journal of Artificial Intelligence Research*, and can be referenced as follows.

Wilson, D. Randall, and Tony R. Martinez, (1997). "Heterogeneous Distance Functions for Instance-Based Learning," *Journal of Artificial Intelligence Research (JAIR)*, vol. 6, no. 1, pp. 1-34.

Chapter 4

Heterogeneous Radial Basis Function Networks

"What goes around, comes around. Unless it's square."
—Steve Cordon.

Proceedings of the International Conference on Neural Networks (ICNN'96),
vol. 2, pp. 1263-1267, June 1996.

Abstract

Radial Basis Function (RBF) networks typically use a distance function designed for numeric attributes, such as Euclidean or city-block distance. This paper presents a heterogeneous distance function which is appropriate for applications with symbolic attributes, numeric attributes, or both. Empirical results on 30 data sets indicate that the heterogeneous distance metric yields significantly improved generalization accuracy over Euclidean distance in most cases involving symbolic 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.

In such cases, neural networks can be used as tools to make reasonable solutions possible or good solutions more economical. Such an automated solution is often more accurate than a hard-coded program, because it learns from actual data instead of making assumptions about the problem. It often can adapt as the domain changes and often takes less time to find a good solution than a programmer would. In addition, inductive learning solutions may generalize well to unforeseen circumstances.

Radial Basis Function (RBF) networks [Chen, Cowen & Grant, 1991; Wasserman, 1993; Wong, 1991] have received much attention recently because they provide accurate generalization on a wide range of applications, yet can often be trained orders of magnitude faster [Renals & Rohwer, 1989] than other models such as backpropagation neural networks [Rumelhart, 1986] or genetic algorithms [Spears et al., 1993].

Radial basis function networks make use of a distance function to find out how different two input vectors are (one being presented to the network and the other stored in a hidden node). This distance function is typically designed for numeric attributes only and is inappropriate for nominal (unordered symbolic) attributes.

This paper introduces a heterogeneous distance function which allows radial basis function networks to appropriately handle applications that contain nominal attributes, numeric attributes, or both. Section 2 introduces the basic radial basis function network that will be used to demonstrate the usefulness of the heterogeneous distance function. Section 3 introduces the distance function itself. Section 4 presents empirical results which indicate that in most cases the heterogeneous distance function significantly improves generalization over Euclidean distance when symbolic attributes are present and never reduces accuracy in completely numeric domains. Section 5 presents conclusions and future research areas.

2. Radial Basis Function Network

This section presents a radial basis function (RBF) network that is used as a probabilistic neural network (PNN) [Specht, 1992] for classification. The distance function presented in this paper could be appropriately used on many different kinds of basis-function networks, so this particular network is just one example of its use. This network was chosen because of its simplicity, which helps to focus on the new distance function instead of on other factors.

The network learns from a *training set* T, which is a collection of examples called *instances*. Each instance i has an input vector $\boldsymbol{y}_i$, and an output class, denoted as $class_i$. During execution, the network receives additional input vectors, denoted as $\boldsymbol{x}$, and outputs the class that $\boldsymbol{x}$ seems most likely to belong to.

The probabilistic neural network is shown in Figure 1. The first (leftmost) layer contains input nodes, each of which receives an input value from the corresponding element in the input vector $\boldsymbol{x}$. Thus, for an application with m input attributes, there are m input nodes. All connections in the network have a weight of 1. In essence, this means that the input vector is passed directly to each hidden node.

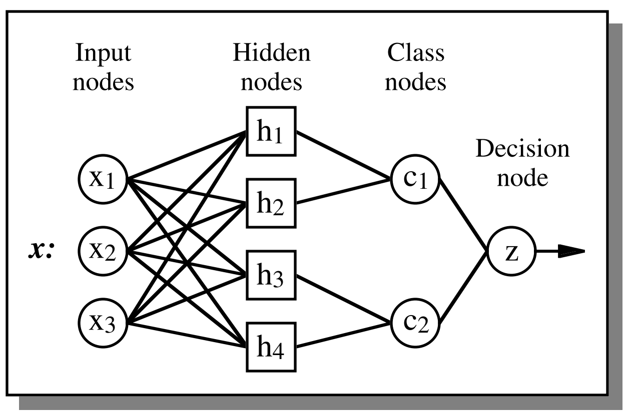


Figure 1. Radial Basis Function Network.

There is one hidden node for each training instance i in the training set. Each hidden node h_i has a center point $\boldsymbol{y}_i$ associated with it, which is the input vector of instance i. A hidden node also has a value σ_i which determines the size of its *receptive field*. This value is set to the distance of the nearest neighbor of i in the training set, using the same distance function as that used during execution.

A hidden node receives an input vector $\boldsymbol{x}$ and outputs an activation given by the function:

$$g(\boldsymbol{x}, \boldsymbol{y}_i, \sigma_i) = \exp[-D^2(\boldsymbol{x}, \boldsymbol{y}_i) / 2\sigma_i^2] \tag{1}$$

where D is a distance function such as Euclidean distance or the heterogeneous distance function that will be discussed in Section 3. This function g is a Gaussian function which returns a value of 1 if $\boldsymbol{x}$ and $\boldsymbol{y}_i$ are equal, and drops to an insignificant value as the distance grows.

Each hidden node h_i is connected to a single class node. If the output class of instance i is j, then h_i is connected to class node c_j. Each *class node* c_j computes the sum of the activations of the hidden nodes that are connected to it (i.e., all the hidden nodes for a particular class) and passes this sum to a decision node. The *decision node* outputs the class with the highest summed activation.

One of the greatest advantages of this network is that it does not require any iterative training. One disadvantage of this network is that it has one hidden node for each training instance and thus requires more computational resources during execution than many other models. In addition, it does not iteratively train weights on any of the connections, which can make its generalization less flexible.

3. Heterogeneous Distance Function

In Section 2, a probabilistic neural network was presented using radial basis functions and a simple weighting scheme that avoided iterative training. In this section, several alternatives for the distance function D are defined, including a new heterogeneous distance function H.

Radial basis functions typically use the *Euclidean distance* function:

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

where m is the number of input variables (attributes) in the application. An alternative function, the *city-block* or *Manhattan* distance function, uses less computation and often does not significantly change the results [Specht, 1992]. It is defined as:

$$M(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i| \tag{3}$$

One problem with both of these distance functions is that they assume that the input variables are linear. However, there are many applications that have *nominal* attributes. A nominal attribute is one with a discrete set of attribute values that are unordered. 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, because numbers assigned to the values are in an arbitrary order. In such cases a distance function is needed that handles nominal inputs appropriately.

Stanfill & Waltz [1986] introduced the *value difference metric* (VDM) which has been used as the basis of several distance functions in the area of machine learning

[Cost & Salzberg, 1993, Domingos, 1995; Rachlin et al., 1994]. Using VDM, the distance between two values *x* and *y* of a single attribute *a* is given as

$$vdm_a(x, y) = \sum_{c=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2 \tag{4}$$

where $N_{a,x}$ is the number of times attribute *a* had value *x*; $N_{a,x,c}$ is the number of times attribute *a* had value *x* and the output class was *c*; and *C* is the number of output classes. Using this distance measure, two values are considered to be closer if they have more similar classifications, regardless of the order of the values.

Some models that have used the VDM or extensions of it (notably PEBLS [Rachlin et al., 1994]) have discretized continuous attributes into a somewhat arbitrary number of discrete ranges and then treated these values as nominal values. Discretization throws away much of the information available to the learning model and often reduces generalization accuracy [Ventura & Martinez, 1995].

The Heterogeneous Radial Basis Function (HRBF) model presented in this paper makes use of a new distance function that uses the above part of the VDM as a building block. In the HRBF model, the heterogeneous distance *H* between two vectors ***x*** and ***y*** is given as

$$H(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{a=1}^{m} d_a(x_a, y_a)^2} \tag{5}$$

$$d_a(x, y) = \begin{cases} 1 & \text{if } x \text{ or } y \text{ is unknown; otherwise...} \\ normalized\_vdm_a(x, y) & \text{if } a \text{ is nominal} \\ normalized\_diff_a(x, y) & \text{if } a \text{ is numeric} \end{cases} \tag{6}$$

where *m* is the number of attributes. The function $d_a(x,y)$ returns a distance between the two attribute values x and y using one of two functions (defined below), depending on whether the attribute is nominal or numeric. Many data sets contain unknown input values which must be handled appropriately in a practical system. The function $d_a(x,y)$ therefore returns a distance of 1 if either *x* or *y* is unknown. Other more complicated methods have been tried, as in [Wilson, 1994], but with little effect on accuracy. The function *H* is similar to that used in [Giraud-Carrier & Martinez, 1995], except that it uses VDM instead of an overlap metric for nominal values and normalizes differently.

One weakness of the basic Euclidean and Manhattan distance functions is that if one of the input variables has a relatively large range, then it can overpower the other input variables. For example, suppose an application has just two input attributes, *f* and *g*. If *f* can have values from 1 to 1000 and *g* has values only from 1 to 10, then *g*'s influence on the distance function will usually be overpowered by *f*'s influence.

Therefore, distances are often *normalized* by dividing the distance for each variable by the range of that attribute, so that the distance for each input variable is in the range 0..1. However, this allows outliers (extreme values) to have a profound effect on the contribution of an attribute. For example, if a variable has values which are in the range 0..10 in almost every case but with one (possibly erroneous) value of 50, then dividing by the range would almost always result in a value less than 0.2. A more robust alternative is to divide the values by the standard deviation in order to reduce the effect of extreme values on the typical cases.

In the heterogeneous distance metric, the situation is more complicated because the nominal and numeric distance values come from different types of measurements. It is therefore necessary to find a way to scale these two different measurements into approximately the same range in order to give each variable a similar influence on the overall distance measurement.

Since 95% of the values in a normal distribution fall within two standard deviations of the mean, the difference between numeric values is divided by 4 standard deviations in order to scale each value into a range that is usually of width 1.0.

Using VDM, the average value for $N_{a,x,c}/N_{a,x}$ (as well as for $N_{a,y,c}/N_{a,y}$) is $1/C$. Since the difference is squared and then added C times, the sum is usually in the neighborhood of $C(1/C^2)=1/C$. This sum is therefore multiplied by C to get it in the range 0..1, making it roughly equal in influence to normalized numeric values.

The functions *normalized_vdm* and *normalized_diff* are thus defined as

$$normalized\_vdm_a(x,y) = \sqrt{C * \sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^2} \qquad (7)$$

$$normalized\_diff_a(x,y) = \frac{|x-y|}{4\sigma_a} \qquad (8)$$

where C is the number of classes, N is defined as in (4), and σ_a is the standard deviation of the numeric values of attribute a. Note that in practice the square root in (7) is not performed since the squared attribute distances are needed in (5) to compute H. Similarly, the square root in (5) is not typically performed in computing H, since the squared distance (H^2 instead of D^2 in this case) is used in (1) to compute g, the activation of a hidden node.

4. Empirical Results

The Heterogeneous Radial Basis Function (HRBF) algorithm was implemented and tested on several databases from the Machine Learning Database Repository at the University of California Irvine [Murphy & Aha, 1993].

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 network using 90% of the training instances in hidden nodes and then using this network to classify the remaining 10% of the instances to see how many were classified correctly.

In order to see what effect the new heterogeneous distance function has on accuracy, a homogeneous version of the algorithm was implemented as well, which is exactly the same as HRBF, except that it uses a normalized Euclidean distance function. This is accomplished by using $normalized\_diff_a(x,y)$ instead of

$normalized\_vdm_a(x,y)$ in (6) for nominal as well as for numeric attributes. The homogeneous algorithm will be referred to as the *default* algorithm, or simply *RBF*. Both algorithms used the same training sets and test sets for each trial.

The average accuracy for each database over all trials is shown in Figure 2. A bold value indicates which value was highest for each database. 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% or higher confidence interval.

Figure 2 also lists the number of continuous and nominal input attributes for each database. Note that the accuracy for every application that has only numeric attributes is exactly the same for both RBF and HRBF. This is no surprise, since the distance functions are equivalent on numeric attributes.

Database	RBF (Euclidean)	HRBF	Numeric Attributes	Nominal Attributes
Annealing	**76.19**	76.06	9	29
Audiology	36.00	**54.00****	0	69
Australian-Credit	80.14	**83.77****	6	8
Bridges	52.36	**55.27***	4	7
Credit-Screening	75.36	**83.48****	6	9
DNA Promoters	54.27	**76.91****	0	57
Echocardiogram	78.04	**79.46**	7	2
Flags	45.74	**57.11****	10	18
Hayes-Roth	52.17	**65.96****	0	4
Heart	**80.74**	80.00	7	6
Heart-Disease (Hungarian)	64.00	**74.92****	7	6
Heart-Disease (More)	45.95	45.95	7	6
Heart-Disease (Swiss)	38.85	38.85	7	6
Hepatitis	79.33	79.33	6	13
Horse-Colic	67.09	67.09	7	16
House-Votes-84	69.22	**79.77****	0	16
Image Segmentation	80.48	80.48	18	1
Solar-Flare 1	**81.71**	81.41	1	9
Solar-Flare 2	99.53	99.53	1	11
Soybean-Large	13.01	**35.10****	6	29
Thyroid-Disease (Euthyroid)	90.74	90.74	7	18
Tic-Tac-Toe	65.78	**79.74****	0	9
Zoo	**78.89****	73.33	0	16
Average:	65.46	**71.23****		
Numeric Databases				
Breast-Cancer-Wisconsin	97.00	97.00	9	0
Liver-Disorders	62.50	62.50	6	0
Iris	94.00	94.00	4	0
Pima-Indians-Diabetes	76.30	76.30	8	0
Sat.Test	85.65	85.65	36	0
Vowel	92.01	92.01	10	0
Wine	94.38	94.38	13	0

Figure 2. Comparative experimental results of RBF and HRBF.

However, on the databases that have some or all nominal attributes, HRBF obtained higher generalization accuracy than RBF in 12 out of 23 cases, 10 of which were significant at the 95% level or above. RBF had a higher accuracy in only four cases, and only one of those (the *Zoo* data set) had a difference that was statistically significant.

It is interesting to note that in the *Zoo* data set, 15 out of 16 of the attributes are boolean, and the remaining attribute, while not linear, is actually an ordered attribute. These attributes are tagged as nominal, but the Euclidean distance function is appropriate for them as well.

In all, HRBF performed as well or better than the default algorithm in 26 out of 30 cases.

The above results indicate that the heterogeneous distance function is typically more appropriate than the Euclidean distance function for applications with one or more nominal attributes, and is equivalent to it for domains without nominal attributes.

5. Conclusions & Future Research

The Heterogeneous Radial Basis Function (HRBF) network uses a normalized heterogeneous distance function which is typically more appropriate than the Euclidean distance function for applications that have at least some nominal or symbolic attributes. By using a more appropriate distance function, higher generalization accuracy can be obtained on most typical heterogeneous or completely symbolic domains. Furthermore, the heterogeneous distance function is equivalent to a normalized Euclidean distance function in completely numeric domains so generalization accuracy will be identical in those domains as well.

In this paper the heterogeneous distance function was used with a probabilistic neural network for classification, which allowed very fast training at the cost of a large, static network. However, this function is appropriate for a wide range of basis function networks that use distance functions.

Current research is seeking to test the heterogeneous distance function on a variety of other models, including various Radial Basis Function networks and instance-based machine learning systems. The normalization factors are also being examined to see if they provide the best possible normalization.

In addition, it appears that some data which is tagged as "nominal" is often somewhat ordered. It is hypothesized that if the values of nominal attributes are randomly rearranged then the HRBF would perform about the same (since it does not depend on the ordering of nominal values), but that the homogeneous RBF would suffer a loss in accuracy. The accuracy of this hypothesis and the severity of the loss in accuracy are currently being explored.

The results of this research are encouraging, and show that heterogeneous distance functions can be used to apply basis function networks to a wider variety of applications and achieve higher generalization accuracy than the homogeneous distance functions used in the past.

References

Chen, S., C. R. N. Cowen, and P. M. Grant, "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 302-309, 1991.

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

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

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

Murphy, P. M., and D. W. Aha, *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: ftp://ftp.ics.uci.edu/pub/machine-learning-databases, 1993.

Rachlin, John, Simon Kasif, Steven Salzberg, David W. Aha, "Towards a Better Understanding of Memory-Based and Bayesian Classifiers," in *Proceedings of the Eleventh International Machine Learning Conference,* New Brunswick, NJ: Morgan Kaufmann, pp. 242-250, 1994.

Renals, Steve, and Richard Rohwer, "Phoneme Classification Experiments Using Radial Basis Functions," *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'89)*, vol. 1, pp. 461-467, 1989.

Rumelhart, D. E., and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, 1986.

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

Specht, Donald F., (1992). "Enhancements to Probabilistic Neural Networks," in *Proceedings International Joint Conference on Neural Networks (IJCNN '92)*, vol. 1, pp. 761-768.

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

Ventura, Dan, and Tony Martinez, "An Empirical Comparison of Discretization Models," *Proceedings of the 10th International Symposium on Computer and Information Sciences*, pp. 443-450, 1995.

Wasserman, Philip D., *Advanced Methods in Neural Computing*, New York, NY: Van Nostrand Reinhold, pp. 147-176, 1993.

Wilson, D. Randall, *Prototype Styles of Generalization*, Master's Thesis, Brigham Young University, 1994.

Wong, Yiu-fai, "How Gaussian Radial Basis Functions Work," *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'91)*, vol. 2, pp. 133-138, 1991.

Chapter 5

Improved Heterogeneous Distance Functions

"Close only counts in horseshoes and thermonuclear warfare."
—Richard A. Ingalls

Journal of Artificial Intelligence Research (JAIR), vol. 6, no. 1, pp. 1-34, 1997.

Early work on this subject was also published in:
Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks (AIE'96), pp. 11-14, 1996.

Abstract

Instance-based learning techniques typically handle continuous and linear input values well, but often do not handle nominal input attributes appropriately. The Value Difference Metric (VDM) was designed to find reasonable distance values between nominal attribute values, but it largely ignores continuous attributes, requiring discretization to map continuous values into nominal values. This paper proposes three new heterogeneous distance functions, called the Heterogeneous Value Difference Metric (HVDM), the Interpolated Value Difference Metric (IVDM), and the Windowed Value Difference Metric (WVDM). These new distance functions are designed to handle applications with nominal attributes, continuous attributes, or both. In experiments on 48 applications the new distance metrics achieve higher classification accuracy on average than three previous distance functions on those datasets that have both nominal and continuous attributes.

1. Introduction

Instance-Based Learning (IBL) [Aha, Kibler & Albert, 1991; Aha, 1992; Wilson & Martinez, 1993; Wettschereck, Aha & Mohri, 1995; Domingos, 1995] is a paradigm of learning in which algorithms typically store some or all of the n available training examples (*instances*) from a *training set*, T, during learning. Each instance has an *input vector* $\boldsymbol{x}$, and an *output class* c. During *generalization*, these systems use a distance function to determine how close a new input vector $\boldsymbol{y}$ is to each stored instance, and use the nearest instance or instances to predict the output class of $\boldsymbol{y}$ (i.e., to *classify* $\boldsymbol{y}$). Some instance-based learning algorithms are referred to as nearest neighbor techniques [Cover & Hart, 1967; Hart, 1968; Dasarathy, 1991], and memory-based reasoning methods [Stanfill & Waltz, 1986; Cost & Salzberg, 1993; Rachlin et al., 1994] overlap significantly with the instance-based paradigm as well. Such algorithms have had much success on a wide variety of *applications* (real-world classification tasks).

Many neural network models also make use of distance functions, including radial basis function networks [Broomhead & Lowe, 1988; Renals & Rohwer, 1989; Wasserman, 1993], counterpropagation networks [Hecht-Nielsen, 1987], ART [Carpenter & Grossberg, 1987], self-organizing maps [Kohonen, 1990] and competitive learning [Rumelhart & McClelland, 1986]. Distance functions are also used in many fields besides machine learning and neural networks, including statistics [Atkeson, Moore & Schaal, 1996], pattern recognition [Diday, 1974; Michalski, Stepp & Diday, 1981], and cognitive psychology [Tversky, 1977; Nosofsky, 1986].

There are many distance functions that have been proposed to decide which instance is closest to a given input vector [Michalski, Stepp & Diday, 1981; Diday, 1974]. Many of these metrics work well for numerical attributes but do not appropriately handle nominal (i.e., discrete, and perhaps unordered) attributes.

The Value Difference Metric (VDM) [Stanfill & Waltz, 1986] was introduced to define an appropriate distance function for nominal (also called *symbolic*) attributes. The Modified Value Difference Metric (MVDM) uses a different weighting scheme than VDM and is used in the PEBLS system [Cost & Salzberg, 1993; Rachlin et al., 1994]. These distance metrics work well in many nominal domains, but they do not handle continuous attributes directly. Instead, they rely upon *discretization* [Lebowitz, 1985; Schlimmer, 1987], which can degrade generalization accuracy [Ventura & Martinez, 1995].

Many real-world applications have both nominal and linear attributes, including, for example, over half of the datasets in the UCI Machine Learning Database Repository [Merz & Murphy, 1996]. This paper introduces three new distance functions that are more appropriate than previous functions for applications with both nominal and continuous attributes. These new distance functions can be incorporated into many of the above learning systems and areas of study, and can be augmented with weighting schemes [Wettschereck, Aha & Mohri, 1995; Atkeson, Moore & Schaal, 1996] and other enhancements that each system provides.

The choice of distance function influences the *bias* of a learning algorithm. A bias is "a rule or method that causes an algorithm to choose one generalized output over another" [Mitchell, 1980]. A learning algorithm must have a bias in order to generalize, and it has been shown that no learning algorithm can generalize more accurately than any other when summed over all possible problems [Schaffer, 1994] (unless information about the problem *other than the training data* is available). It follows then that no distance function can be strictly better than any other in terms of generalization ability, when considering all possible problems with equal probability.

However, when there is a higher probability of one class of problems occurring than another, some learning algorithms can generalize more accurately than others [Wolpert, 1993]. This is not because they are better when summed over all problems, but because the problems on which they perform well are more likely to occur. In this sense, one algorithm or distance function can be an improvement over another in that it has a higher probability of good generalization than another, because it is better matched to the kinds of problems that will likely occur.

Many learning algorithms use a *bias of simplicity* [Mitchell, 1980; Wolpert, 1993] to generalize, and this bias is *appropriate*—meaning that it leads to good generalization accuracy—for a wide variety of real-world applications, though the meaning of *simplicity* varies depending upon the representational language of each learning algorithm. Other biases, such as decisions made on the basis of additional domain knowledge for a particular problem [Mitchell, 1980], can also improve generalization.

In this light, the distance functions presented in this paper are more appropriate than those used for comparison in that they on average yield improved generalization accuracy on a collection of 48 applications. The results are theoretically limited to this set of datasets, but the hope is that these datasets are representative of other problems that will be of interest (and occur frequently) in the real world, and that the distance functions presented here will be useful in such cases, especially those involving both continuous and nominal input attributes.

Section 2 provides background information on distance functions used previously. Section 3 introduces a distance function that combines Euclidean distance and VDM to handle both continuous and nominal attributes. Sections 4 and 5 present two extensions of the Value Difference Metric which allow for direct use of continuous attributes. Section 4 introduces the Interpolated Value Difference Metric (IVDM), which uses interpolation of probabilities to avoid problems related to discretization. Section 5 presents the Windowed Value Difference Metric (WVDM), which uses a more detailed probability density function for a similar interpolation process.

Section 6 presents empirical results comparing three commonly used distance functions with the three new functions presented in this paper. The results are obtained from using each of the distance functions in an instance-based learning system on 48 datasets. The results indicate that the new heterogeneous distance functions are more appropriate than previously used functions on datasets with both nominal and linear attributes, in that they achieve higher average generalization accuracy on these datasets. Section 7 discusses related work, and Section 8 provides conclusions and future research directions.

2. Previous Distance Functions

As mentioned in the introduction, there are many learning systems that depend upon a good distance function to be successful. A variety of distance functions are available for such uses, including the Minkowsky [Batchelor, 1978], Mahalanobis [Nadler & Smith, 1993], Camberra, Chebychev, Quadratic, Correlation, and Chi-square distance metrics [Michalski, Stepp & Diday, 1981; Diday, 1974]; the Context-Similarity measure [Biberman, 1994]; the Contrast Model [Tversky, 1977]; hyperrectangle distance functions [Salzberg, 1991; Domingos, 1995] and others. Several of these functions are defined in Figure 1.

Although there have been many distance functions proposed, by far the most commonly used is the Euclidean Distance function, which is defined as

$$E(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{a=1}^{m}(x_a - y_a)^2} \quad (1)$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are two input vectors (one typically being from a stored instance, and the other an input vector to be classified) and m is the number of input variables (*attributes*) in the application. The square root is often not computed in practice, because the closest instance(s) will still be the closest, regardless of whether the square root is taken.

An alternative function, the *city-block* or *Manhattan* distance function, requires less computation and is defined as

$$M(\boldsymbol{x},\boldsymbol{y}) = \sum_{a=1}^{m}\left|x_a - y_a\right| \quad (2)$$

The Euclidean and Manhattan distance functions are equivalent to the Minkowskian r-distance function [Batchelor, 1978] with $r = 2$ and 1, respectively.

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T Q(\boldsymbol{x}-\boldsymbol{y}) = \sum_{j=1}^{m} \left(\sum_{i=1}^{m} (x_i - y_i) q_{ji} \right) (x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m} (\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}} V^{-1} (\boldsymbol{x}-\boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i} \left(\frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}} \right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} \mathrm{sign}(x_i - x_j)\,\mathrm{sign}(y_i - y_j)$$

Figure 1. Equations of selected distance functions ($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

2.1. Normalization

One weakness of the basic Euclidean distance function is that if one of the input attributes has a relatively large range, then it can overpower the other attributes. For example, if an application has just two attributes, A and B, and A can have values from 1 to 1000, and B has values only from 1 to 10, then B's influence on the distance function will usually be overpowered by A's influence. Therefore, distances are often *normalized* by dividing the distance for each attribute by the *range* (i.e., maximum-minimum) of that attribute, so that the distance for each attribute is in the approximate range 0..1. In order to avoid outliers, it is also common to divide by the standard deviation instead of range, or to "trim" the range by removing the highest and lowest few percent (e.g., 5%) of the data from consideration in defining the range. It is also possible to map any value outside this range to the minimum or maximum value to avoid normalized values outside the range 0..1. Domain knowledge can often be used to decide which method is most appropriate.

Related to the idea of normalization is that of using attribute weights and other weighting schemes. Many learning systems that use distance functions incorporate various weighting schemes into their distance calculations [Wettschereck, Aha & Mohri, 1995; Atkeson, Moore & Schaal, 1996]. The improvements presented in this paper are independent of such schemes, and most of the various weighting schemes (as well as other enhancements such as instance pruning techniques) can be used in conjunction with the new distance functions presented here.

2.2. Attribute Types

None of the distance functions shown in Figure 1, including Euclidean distance, appropriately handle non-continuous input attributes.

An attribute can be linear or nominal, and a linear attribute can be continuous or discrete. A *continuous* (or *continuously valued*) attribute uses real values, such as the mass of a planet or the velocity of an object. A *linear discrete* (or *integer*) attribute can have only a discrete set of linear values, such as *number of children*.

It can be argued that any value stored in a computer is discrete at some level. The reason continuous attributes are treated differently is that they can have so many different values that each value may appear only rarely (perhaps only once in a particular application). This causes problems for algorithms such as VDM (described in Section 2.4) that depend on testing two values for equality, because two continuous values will rarely be equal, though they may be quite close to each other.

A *nominal* (or *symbolic*) attribute is a discrete attribute whose values are not necessarily in any linear order. For example, a variable representing color might have values such as *red, green, blue, brown, black* and *white*, which could be represented by the integers 1 through 6, respectively. Using a linear distance measurement such as (1) or (2) on such values makes little sense in this case.

2.3. Heterogeneous Overlap-Euclidean Metric (HOEM)

One way to handle applications with both continuous and nominal attributes is to use a heterogeneous distance function that uses different attribute distance functions on different kinds of attributes. One approach that has been used is to use the *overlap* metric for nominal attributes and normalized Euclidean distance for linear attributes.

For the purposes of comparison during testing, we define a heterogeneous distance function that is similar to that used by IB1, IB2 and IB3 [Aha, Kibler & Albert, 1991; Aha, 1992] as well as that used by Giraud-Carrier & Martinez [1995]. This function defines the distance between two values x and y of a given attribute a as

$$d_a(x,y) = \begin{cases} 1, & \text{if } x \text{ or } y \text{ is unknown, else} \\ overlap(x,y), & \text{if } a \text{ is nominal, else} \\ rn\_diff_a(x,y) & \end{cases} \tag{3}$$

Unknown attribute values are handled by returning an attribute distance of 1 (i.e., a maximal distance) if either of the attribute values is unknown. The function *overlap* and the range-normalized difference *rn_diff* are defined as

$$overlap(x,y) = \begin{cases} 0, & \text{if } x = y \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

$$rn\_diff_a(x,y) = \frac{|x-y|}{range_a} \tag{5}$$

The value $range_a$ is used to normalize the attributes, and is defined as

$$range_a = max_a - min_a \tag{6}$$

where max_a and min_a are the maximum and minimum values, respectively, observed in the training set for attribute a. 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, such cases are rare, and when they do occur, a large difference may be acceptable anyway. The normalization serves to scale the attribute down to the point where differences are almost always less than one.

The above definition for d_a 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 by the Heterogeneous Overlap-Euclidean Metric function $HOEM(\boldsymbol{x},\boldsymbol{y})$

$$HOEM(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{a=1}^{m} d_a(x_a, y_a)^2} \tag{7}$$

This distance function removes the effects of the arbitrary ordering of nominal values, but its overly simplistic approach to handling nominal attributes fails to make use of additional information provided by nominal attribute values that can aid in generalization.

2.4. Value Difference Metric (VDM)

The Value Difference Metric (VDM) was introduced by Stanfill and Waltz [1986] to provide an appropriate distance function for nominal attributes. A simplified version of the VDM (without the weighting schemes) defines the distance between two values x and y of an attribute a as

$$vdm_a(x,y) = \sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^q = \sum_{c=1}^{C} \left| P_{a,x,c} - P_{a,y,c} \right|^q \tag{8}$$

where

- $N_{a,x}$ is the number of instances in the training set T that have value x for attribute a;
- $N_{a,x,c}$ is the number of instances in T that have value x for attribute a and output class c;

- C is the number of output classes in the problem domain;
- q is a constant, usually 1 or 2; and
- $P_{a,x,c}$ is the conditional probability that the output class is c given that attribute a has the value x, i.e., $P(c \mid x_a)$. As can be seen from (8), $P_{a,x,c}$ is defined as

$$P_{a,x,c} = \frac{N_{a,x,c}}{N_{a,x}} \tag{9}$$

where $N_{a,x}$ is the sum of $N_{a,x,c}$ over all classes, i.e.,

$$N_{a,x} = \sum_{c=1}^{C} N_{a,x,c} \tag{10}$$

and the sum of $P_{a,x,c}$ over all C classes is 1 for a fixed value of a and x.

Using the distance measure $vdm_a(x,y)$, two values are considered to be closer if they have more similar classifications (i.e., more similar correlations with the output classes), regardless of what order the values may be given in. In fact, linear discrete attributes can have their values remapped randomly without changing the resultant distance measurements.

For example, if an attribute *color* has three values *red, green* and *blue*, and the application is to identify whether or not an object is an apple, *red* and *green* would be considered closer than *red* and *blue* because the former two both have similar correlations with the output class *apple*.

The original VDM algorithm [Stanfill & Waltz, 1986] makes use of feature weights that are not included in the above equations, and some variants of VDM [Cost & Salzberg, 1993; Rachlin et al., 1994; Domingos, 1995] have used alternate weighting schemes. As discussed earlier, the new distance functions presented in this paper are independent of such schemes and can in most cases make use of similar enhancements.

One problem with the formulas presented above is that they do not define what should be done when a value appears in a new input vector that never appeared in the training set. If attribute a never has value x in any instance in the training set, then $N_{a,x,c}$ for all c will be 0, and $N_{a,x}$ (which is the sum of $N_{a,x,c}$ over all classes) will also be 0. In such cases $P_{a,x,c} = 0/0$, which is undefined. For nominal attributes, there is no way to know what the probability should be for such a value, since there is no inherent ordering to the values. In this paper we assign $P_{a,x,c}$ the default value of 0 in such cases (though it is also possible to let $P_{a,x,c} = 1/C$, where C is the number of output classes, since the sum of $P_{a,x,c}$ for $c = 1..C$ is always 1.0).

If this distance function is used directly on continuous attributes, the values can all potentially be unique, in which case $N_{a,x}$ is 1 for every value x, and $N_{a,x,c}$ is 1 for one value of c and 0 for all others for a given value x. In addition, new vectors are likely to have unique values, resulting in the division by zero problem above. Even if the value of 0 is substituted for 0/0, the resulting distance measurement is nearly useless.

Even if all values are not unique, there are often enough different values for a continuous attribute that the statistical sample is unreliably small for each value, and the distance measure is still untrustworthy. Because of these problems, it is inappropriate to use the VDM directly on continuous attributes.

2.5. Discretization

One approach to the problem of using VDM on continuous attributes is *discretization* [Lebowitz, 1985; Schlimmer, 1987; Ventura, 1995]. Some models that have used the VDM or variants of it [Cost & Salzberg, 1993; Rachlin et al., 1994; Mohri & Tanaka, 1994] have discretized continuous attributes into a somewhat arbitrary number of discrete ranges, and then treated these values as nominal (discrete unordered) values. This method has the advantage of generating a large enough statistical sample for each nominal value that the P values have some significance. However, discretization can lose much of the important information available in the continuous values. For example, two values in the same discretized range are considered equal even if they are on opposite ends of the range. Such effects can reduce generalization accuracy [Ventura & Martinez, 1995].

In this paper we propose three new alternatives, which are presented in the following three sections. Section 3 presents a heterogeneous distance function that uses Euclidean distance for linear attributes and VDM for nominal attributes. This method requires careful attention to the problem of normalization so that neither nominal nor linear attributes are regularly given too much weight.

In Sections 4 and 5 we present two distance functions, the Interpolated Value Difference Metric (IVDM) and the Windowed Value Difference Metric (WVDM), which use discretization to collect statistics and determine values of $P_{a,x,c}$ for continuous values occurring in the training set instances, but then retain the continuous values for later use. During generalization, the value of $P_{a,y,c}$ for a continuous value y is interpolated between two other values of P, namely, $P_{a,x1,c}$ and $P_{a,x2,c}$, where $x_1 \le y \le x_2$. IVDM and WVDM are essentially different techniques for doing a nonparametric probability density estimation [Tapia & Thompson, 1978] to determine the values of P for each class. A generic version of the VDM algorithm, called the *discretized value difference metric* (DVDM) is used for comparisons with the two new algorithms.

3. Heterogeneous Value Difference Metric (HVDM)

As discussed in the previous section, the Euclidean distance function is inappropriate for nominal attributes, and VDM is inappropriate for continuous attributes, so neither is sufficient on its own for use on a heterogeneous application, i.e., one with both nominal and continuous attributes.

In this section, we define a heterogeneous distance function *HVDM* that returns the distance between two input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. It is defined as follows

$$HVDM(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{a=1}^{m} d_a(x_a, y_a)^2} \tag{11}$$

where m is the number of attributes. The function $d_a(x,y)$ returns a distance between the two values x and y for attribute a and is defined as

$$d_a(x,y) = \begin{cases} 1, & \text{if } x \text{ or } y \text{ is unknown; otherwise...} \\ normalized\_vdm_a(x,y), & \text{if } a \text{ is nominal} \\ normalized\_diff_a(x,y), & \text{if } a \text{ is linear} \end{cases} \quad (12)$$

The function $d_a(x,y)$ uses one of two functions (defined below in Section 3.1), depending on whether the attribute is nominal or linear. Note that in practice the square root in (11) is not typically performed because the distance is always positive, and the nearest neighbor(s) will still be nearest whether or not the distance is squared. However, there are some models (e.g., distance-weighted k-nearest neighbor, [Dudani, 1976]] that require the square root to be evaluated.

Many applications contain unknown input values which must be handled appropriately in a practical system [Quinlan, 1989]. The function $d_a(x,y)$ therefore returns a distance of 1 if either x or y is unknown, as is done by Aha, Kibler & Albert [1991] and Giraud-Carrier & Martinez [1995]. Other more complicated methods have been tried [Wilson & Martinez, 1993], but with little effect on accuracy.

The function HVDM is similar to the function HOEM given in Section 2.3, except that it uses VDM instead of an overlap metric for nominal values and it also normalizes differently. It is also similar to the distance function used by RISE 2.0 [Domingos, 1995], but has some important differences noted below in Section 3.2.

Section 3.1 presents three alternatives for normalizing the nominal and linear attributes. Section 3.2 presents experimental results which show that one of these schemes provides better normalization than the other two on a set of several datasets. Section 3.3 gives empirical results comparing HVDM to two commonly used distance functions. [The appendix at the end of this chapter gives recent developments on handling skewed class distributions with HVDM.]

3.1. Normalization

As discussed in Section 2.1, distances are often *normalized* by dividing the distance for each variable by the range of that attribute, so that the distance for each input variable is in the range 0..1. This is the policy used by HOEM in Section 2.3. However, dividing by the range allows outliers (extreme values) to have a profound effect on the contribution of an attribute. For example, if a variable has values which are in the range 0..10 in almost every case but with one exceptional (and possibly erroneous) value of 50, then dividing by the range would almost always result in a value less than 0.2. A more robust alternative in the presence of outliers is to divide the values by the standard deviation to reduce the effect of extreme values on the typical cases.

For the new heterogeneous distance metric HVDM, the situation is more complicated because the nominal and numeric distance values come from different types of measurements: numeric distances are computed from the difference between two linear values, normalized by standard deviation, while nominal attributes are computed from a sum of C differences of probability values (where C is the number of output classes). It is therefore necessary to find a way to scale these two different kinds of measurements into approximately the same range to give each variable a similar influence on the overall distance measurement.

Since 95% of the values in a normal distribution fall within two standard deviations of the mean, the difference between numeric values is divided by 4 standard deviations to scale each value into a range that is usually of width 1. The function *normalized_diff* is therefore defined as shown below in Equation 13.

$$normalized\_diff_a(x,y) = \frac{|x-y|}{4\sigma_a} \tag{13}$$

where σ_a is the standard deviation of the numeric values of attribute a.

Three alternatives for the function *normalized_vdm* were considered for use in the heterogeneous distance function. These are labeled N1, N2 and N3, and the definitions of each are given below.

$$\text{N1: } normalized\_vdm1_a(x,y) = \sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right| \tag{14}$$

$$\text{N2: } normalized\_vdm2_a(x,y) = \sqrt{\sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^2} \tag{15}$$

$$\text{N3: } normalized\_vdm3_a(x,y) = \sqrt{C * \sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^2} \tag{16}$$

The function N1 is Equation (8) with q=1. This is similar to the formula used in PEBLS [Rachlin et al., 1994] and RISE [Domingos, 1995] for nominal attributes.

N2 uses q=2, thus squaring the individual differences. This is analogous to using Euclidean distance instead of Manhattan distance. Though slightly more expensive computationally, this formula was hypothesized to be more robust than N1 because it favors having all of the class correlations fairly similar rather than having some very close and some very different. N1 would not be able to distinguish between these two. In practice the square root is not taken, because the individual attribute distances are themselves squared by the HVDM function.

N3 is the function used in Heterogeneous Radial Basis Function Networks [Wilson & Martinez, 1996], where HVDM was first introduced.

3.2. Normalization Experiments

In order to determine whether each normalization scheme N1, N2 and N3 gave unfair weight to either nominal or linear attributes, experiments were run on 15 databases from the machine learning database repository at the University of California, Irvine [Merz & Murphy, 1996]. All of the datasets for this experiment have at least some nominal and some linear attributes, and thus require a heterogeneous distance function.

In each experiment, five-fold cross validation was used. For each of the five trials, the distance between each instance in the test set and each instance in the training set was computed. When computing the distance for each attribute, the *normalized_diff* function was used for linear attributes, and the *normalized_vdm* function N1, N2, or N3 was used (in each of the three respective experiments) for nominal attributes.

The average distance (i.e., sum of all distances divided by number of comparisons) was computed for each attribute. The average of all the linear attributes for each database was computed and these averages are listed under the heading "avgLin" in Table 1.

Database	avgLin	N1 avgNom	N2 avgNom	N3 avgNom	#Nom.	#Lin.	#C
Anneal	0.427	0.849	0.841	0.859	29	9	6
Australian	0.215	0.266	0.188	0.266	8	6	2
Bridges	0.328	0.579	0.324	0.808	7	4	7
Crx	0.141	0.268	0.193	0.268	9	6	2
Echocardiogram	0.113	0.487	0.344	0.487	2	7	2
Flag	0.188	0.372	0.195	0.552	18	10	8
Heart	0.268	0.323	0.228	0.323	6	7	2
Heart.Cleveland	0.271	0.345	0.195	0.434	6	7	5
Heart.Hungarian	0.382	0.417	0.347	0.557	6	7	5
Heart.Long-Beach-VA	0.507	0.386	0.324	0.417	6	7	5
Heart.More	0.360	0.440	0.340	0.503	6	7	5
Heart.Swiss	0.263	0.390	0.329	0.421	6	7	5
Hepatitis	0.271	0.205	0.158	0.205	13	6	2
Horse-Colic	0.444	0.407	0.386	0.407	16	7	2
Soybean-Large	0.309	0.601	0.301	0.872	29	6	19
Average	**0.299**	**0.422**	**0.313**	**0.492**	11	7	5

Table 1. Average attribute distance for linear and nominal attributes.

The averages of all the nominal attributes for each of the three normalization schemes are listed under the headings "avgNom" in Table 1 as well. The average distance for linear variables is exactly the same regardless of whether N1, N2 or N3 is used, so this average is given only once. Table 1 also lists the number of nominal ("#Nom.") and number of linear ("#Lin.") attributes in each database, along with the number of output classes ("#C").

As can be seen from the overall averages in the first four columns of the last row of Table 1, N2 is closer than N1 or N3. However, it is important to understand the reasons behind this difference in order to know if the normalization scheme N2 will be more robust in general.

Figures 2-4 graphically display the averages shown in Table 1 under the headings N1, N2 and N3, respectively, ordered from left to right by the number of output classes. We hypothesized that as the number of output classes grows, the normalization would get worse for N3 if it was indeed not appropriate to add the scaling factor *C* to the sum. The length of each line indicates how much difference there is between the average distance for nominal attributes and linear attributes. An ideal normalization scheme would have a difference of zero, and longer lines indicate worse normalization.

As the number of output classes grows, the difference for N3 between the linear distances and the nominal distances grows wider in most cases. N2, on the other hand, seems to remain quite close independent of the number of output classes. Interestingly, N1 does almost as poorly as N3, even though it does not use the scaling factor *C*. Apparently the squaring factor provides for a more well-rounded distance metric on nominal attributes similar to that provided by using Euclidean distance instead of Manhattan distance on linear attributes.

The underlying hypothesis behind performing normalization is that proper normalization will typically improve generalization accuracy. A nearest neighbor classifier (with $k=1$) was implemented using HVDM as the distance metric. The system was tested on the heterogeneous datasets appearing in Table 1 using the three different normalization schemes discussed above, using ten-fold cross-validation [Schaffer, 1993], and the results are summarized in Table 2. All the normalization schemes used the same training sets and test sets for each trial. Bold entries indicate which scheme had the highest accuracy. An asterisk indicates that the difference was greater that 1% over the next highest scheme.

As can be seen from the table, the normalization scheme N2 had the highest accuracy, and N1 was substantially lower than the other two. N2 and N3 each had the highest accuracy for 8 domains. More significantly, N2 was over 1% higher 5 times compared to N1 being over 1% higher on just one dataset. N3 was higher than the other two on just one dataset, and had a lower average accuracy than N2.

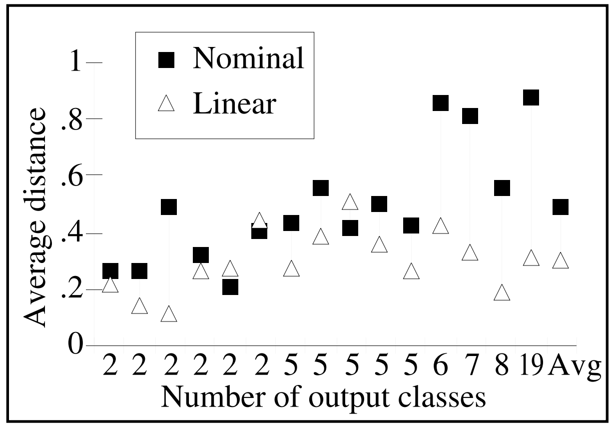


Figure 2. Average distances for N1.

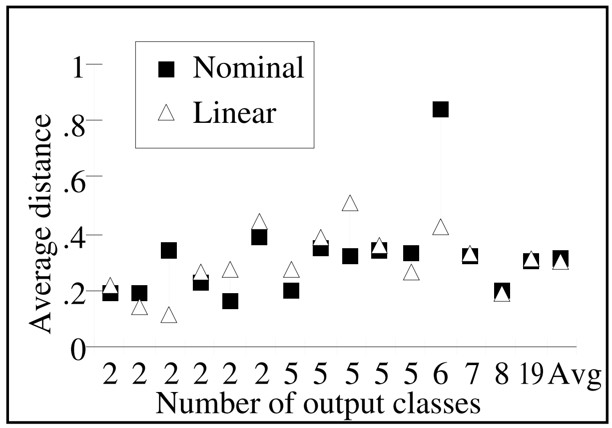


Figure 3. Average distances for N2.

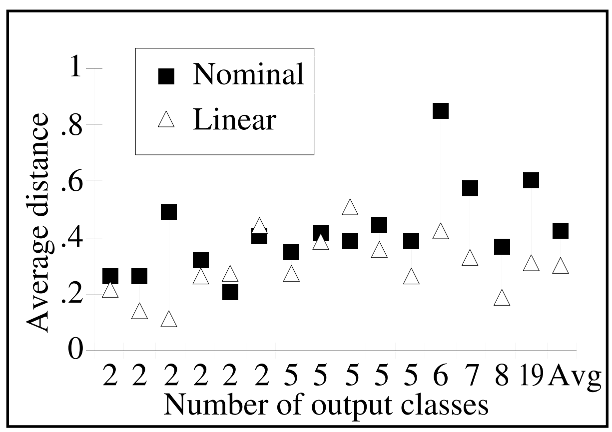


Figure 4. Average distances for N3.

These results support the hypothesis that the normalization scheme N2 achieves higher generalization accuracy than N1 or N3 (on these datasets) due to its more robust normalization though accuracy for N3 is almost as good as N2.

Note that proper normalization will not always necessarily improve generalization accuracy. If one attribute is more important than the others in classification, then giving it a higher weight may improve classification. Therefore, if a more important attribute is given a higher weight accidentally by poor normalization, it may actually

Database	N1	N2	N3
Anneal	93.98	94.61	**94.99**
Australian	71.30	81.45	**81.59**
Bridges	43.36	**59.64**	59.55
Crx	70.29	80.87	**81.01**
Echocardiogram	70.36	**94.82**	**94.82**
Flag	28.95	**55.82***	51.50
Heart.Cleveland	73.88	**76.56***	71.61
Heart.Hungarian	70.75	**76.85***	75.82
Heart.Long-Beach-Va	65.50	65.50	**70.00***
Heart.More	60.03	72.09	**72.48**
Heart	88.46	**89.49**	**89.49**
Heart.Swiss	74.81	**78.52***	75.19
Hepatitis	73.50	76.67	**77.33**
Horse-Colic	**64.75***	60.53	60.53
Soybean-Large	41.45	**90.88***	87.89
Average	66.09	**76.95**	76.25

Table 2. Generalization accuracy using N1, N2 and N3.

improve generalization accuracy. However, this is a random improvement that is not typically the case. Proper normalization should improve generalization in more cases than not when used in typical applications.

As a consequence of the above results, N2 is used as the normalization scheme for HVDM, and the function *normalized_vdm* is defined as in (15).

3.3. Empirical Results of HVDM vs. Euclidean and HOEM

A nearest neighbor classifier (with k=1) using the three distance functions listed in Table 3 was tested on 48 datasets from the UCI machine learning database repository. Of these 48 datasets, the results obtained on the 35 datasets that have at least some nominal attributes are shown in Table 3.

The results are approximately equivalent on datasets with only linear attributes, so the results on the remaining datasets are not shown here, but can be found in Section 6. 10-fold cross-validation was again used, and all three distance metrics used the same training sets and test sets for each trial.

The results of these experiments are shown in Table 3. The first column lists the name of the database (".test" means the database was originally meant to be used as a test set, but was instead used in its entirety as a separate database). The second column shows the results obtained when using the Euclidean distance function normalized by standard deviation on all attributes, including nominal attributes. The next column shows the generalization accuracy obtained by using the HOEM metric, which uses range-normalized Euclidean distance for linear attributes and the overlap metric for nominal attributes. The final column shows the accuracy obtained by using the HVDM distance function which uses the standard-deviation-normalized Euclidean distance (i.e., *normalized_diff* as defined in Equation 13) on linear attributes and the *normalized_vdm* function N2 on nominal attributes.

The highest accuracy obtained for each database is shown in bold. Entries in the Euclid. and HOEM columns that are significantly higher than HVDM (at a 90% or higher confidence level, using a two-tailed paired *t* test) are marked with an asterisk (*). Entries that are significantly lower than HVDM are marked with a "less-than" sign (<).

Database	Euclid.	HOEM	HVDM
Anneal	**94.99**	94.61	94.61
Audiology	60.50 <	72.00 <	**77.50**
Audiology.Test	41.67 <	75.00	**78.33**
Australian	80.58	81.16	**81.45**
Bridges	58.64	53.73	**59.64**
Crx	78.99	**81.01**	80.87
Echocardiogram	**94.82**	**94.82**	**94.82**
Flag	48.95 <	48.84	**55.82**
Heart.Cleveland	73.94	74.96	**76.56**
Heart.Hungarian	73.45 <	74.47	**76.85**
Heart.Long-Beach-Va	**71.50**	71.00 *	65.50
Heart.More	**72.09**	71.90	**72.09**
Heart.Swiss	**93.53** *	91.86	89.49
Hepatitis	**77.50**	77.50	76.67
Horse-Colic	**65.77**	60.82	60.53
House-Votes-84	93.12 <	93.12 <	**95.17**
Image.Segmentation	92.86	**93.57**	92.86
Led+17	42.90 <	42.90 <	**60.70**
Led-Creator	**57.20** *	**57.20** *	56.40
Monks-1.Test	**77.08**	69.43	68.09
Monks-2.Test	59.04 <	54.65 <	**97.50**
Monks-3.Test	87.26 <	78.49 <	**100.00**
Mushroom	**100.00**	**100.00**	**100.00**
Promoters	73.73 <	82.09 <	**92.36**
Soybean-Large	87.26 <	89.20	**90.88**
Soybean-Small	**100.00**	**100.00**	**100.00**
Thyroid.Allbp	94.89	94.89	**95.00**
Thyroid.Allhyper	**97.00**	**97.00**	96.86
Thyroid.Allhypo	**90.39**	**90.39** *	90.29
Thyroid.Allrep	**96.14**	**96.14**	96.11
Thyroid.Dis	**98.21**	**98.21**	**98.21**
Thyroid.Hypothyroid	**93.42**	**93.42**	93.36
Thyroid.Sick-Euthyroid	**68.23**	**68.23**	**68.23**
Thyroid.Sick	**86.93** *	86.89 *	86.61
Zoo	97.78	94.44	**98.89**
Average:	79.44	80.11	**83.38**

Table 3. Generalization accuracy of the Euclidean, HOEM, and HVDM distance functions.

As can be seen from Table 3, the HVDM distance function's overall average accuracy was higher than that of the other two metrics by over 3%. HVDM achieved as high or higher generalization accuracy than the other two distance functions in 21 of the 35 datasets. The Euclidean distance function was highest in 18 datasets, and HOEM was highest in only 12 datasets.

HVDM was significantly higher than the Euclidean distance function on 10 datasets, and significantly lower on only 3. Similarly, HVDM was higher than HOEM on 6 datasets, and significantly lower on only 4.

These results support the hypothesis that HVDM handles nominal attributes more appropriately than Euclidean distance or the heterogeneous Euclidean-overlap metric, and thus tends to achieve higher generalization accuracy on typical applications.

4. Interpolated Value Difference Metric (IVDM)

In this section and Section 5 we introduce distance functions that allow VDM to be applied directly to continuous attributes. This alleviates the need for normalization between attributes. It also in some cases provides a better measure of distance for continuous attributes than linear distance.

For example, consider an application with an input attribute *height* and an output class that indicates whether a person is a good candidate to be a fighter pilot in a particular airplane. Those individuals with heights significantly below *or* above the preferred height might both be considered poor candidates, and thus it could be beneficial to consider their heights as more similar to each other than to those of the preferred height, even though they are farther apart in a linear sense.

On the other hand, linear attributes for which linearly distant values tend to indicate different classifications should also be handled appropriately. The Interpolated Value Difference Metric (IVDM) handles both of these situations, and handles heterogeneous applications robustly.

A generic version of the VDM distance function, called the *discretized value difference metric* (DVDM) will be used for comparisons with extensions of VDM presented in this paper.

4.1. IVDM Learning Algorithm

The original value difference metric (VDM) uses statistics derived from the training set instances to determine a probability $P_{a,x,c}$ that the output class is c given the input value x for attribute a.

When using IVDM, continuous values are discretized into s equal-width intervals (though the continuous values are also retained for later use), where s is an integer supplied by the user. Unfortunately, there is currently little guidance on what value of s to use. A value that is too large will reduce the statistical strength of the values of P, while a value too small will not allow for discrimination among classes. For the purposes of this paper, we use a heuristic to determine s automatically: let s be 5 or C, whichever is greatest, where C is the number of output classes in the problem domain. Current research is examining more sophisticated techniques for determining good values of s, such as cross-validation, or other statistical methods [e.g., Tapia & Thompson, 1978, p. 67]. (Early experimental results indicate that the value of s may not be critical as long as $s \geq C$ and $s \ll n$, where n is the number of instances in the training set.)

The width w_a of a discretized interval for attribute a is given by

$$w_a = \frac{|max_a - min_a|}{s} \tag{17}$$

where max_a and min_a are the maximum and minimum value, respectively, occurring in the training set for attribute a.

As an example, consider the *Iris* database from the UCI machine learning databases. The *Iris* database has four continuous input attributes, the first of which is *sepal length*. Let T be a training set consisting of 90% of the 150 available training instances, and S be a test set consisting of the remaining 10%.

In one such division of the training set, the values in T for the *sepal length* attribute ranged from 4.3 to 7.9. There are only three output classes in this database, so we let s=5, resulting in a width of $|7.9 - 4.3| / 5 = 0.72$. Note that since the discretization is part of the learning process, it would be unfair to use any instances in the test set to help determine how to discretize the values. The discretized value v of a continuous value x for attribute a is an integer from 1 to s, and is given by

$$v = discretize_a(x) = \begin{cases} x, \text{ if } a \text{ is discrete, else} \\ s, \text{ if } x = \max_a, \text{ else} \\ \lfloor (x - \min_a) / w_a \rfloor + 1 \end{cases} \quad (18)$$

After deciding upon s and finding w_a, the discretized values of continuous attributes can be used just like discrete values of nominal attributes in finding $P_{a,x,c}$. Figure 5 lists pseudo-code for how this is done.

```
LearnP(training set T)
    For each attribute a
        For each instance i in T
            Let x be the input value for attribute a of instance i.
            v = discretize_a(x) [which is just x if a is discrete]
            Let c be the output class of instance i.
            Increment N_a,v,c by 1.
            Increment N_a,v by 1.
        For each discrete value v (of attribute a)
            For each class c
                If N_a,v=0
                        Then P_a,v,c=0
                Else P_a,v,c = N_a,v,c / N_a,v
    Return 3-D array P_a,v,c.
```

Figure 5. Pseudo code for finding $P_{a,x,c}$.

For the first attribute of the *Iris* database, the values of $P_{a,x,c}$ are displayed in Figure 6. For each of the five discretized ranges of x, the probability for each of the three corresponding output classes are shown as the bar heights. Note that the heights of the three bars sum to 1.0 for each discretized range. The bold integers indicate the discretized value of each range. For example, a sepal length greater than or equal to 5.74 but less than 6.46 would have a discretized value of 3.

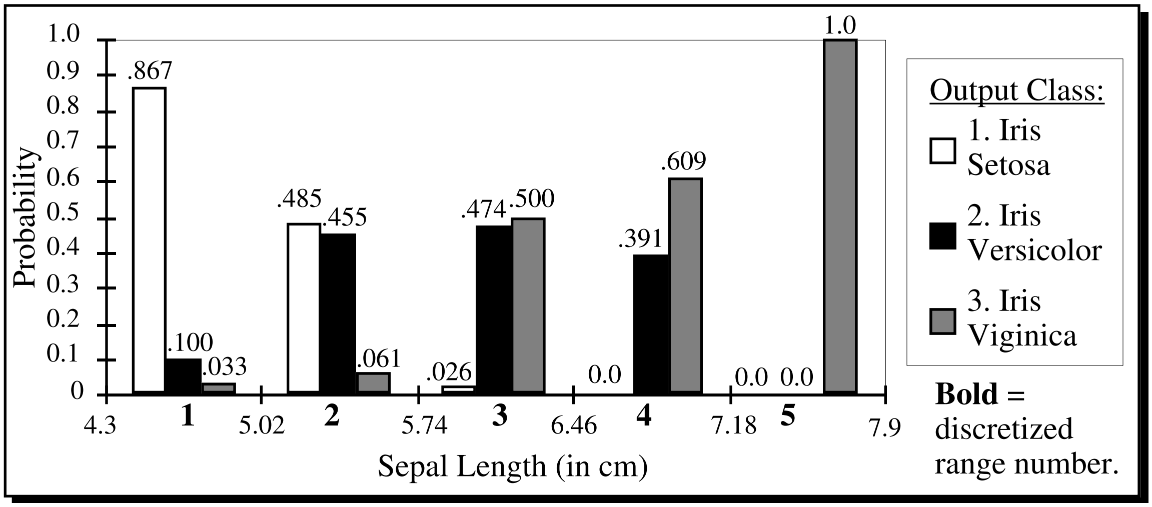


Figure 6. $P_{a,x,c}$ for a=1, x=1..5, c=1..3, on the first attribute of the *Iris* database.

4.2. IVDM and DVDM Generalization

Thus far the DVDM and IVDM algorithms learn identically. However, at this point the DVDM algorithm need not retain the original continuous values because it will use only the discretized values during generalization. On the other hand, the IVDM will use the continuous values.

During generalization, an algorithm such as a nearest neighbor classifier can use the distance function DVDM, which is defined as follows

$$DVDM(\boldsymbol{x},\boldsymbol{y}) = \sum_{a=1}^{m} \left| vdm_a(discretize_a(x_a), discretize_a(y_a)) \right|^2 \tag{19}$$

where $discretize_a$ is as defined in Equation (18) and vdm_a is defined as in Equation (8), with q=2. We repeat it here for convenience

$$vdm_a(x,y) = \sum_{c=1}^{C} \left| P_{a,x,c} - P_{a,y,c} \right|^2 \tag{20}$$

Unknown input values [Quinlan, 1989] are treated as simply another discrete value, as was done in [Domingos, 1995].

	Input Attributes 1	2	3	4		Output Class
A:	5.0	3.6	1.4	0.2	->	1 (Iris Setosa)
B:	5.7	2.8	4.5	1.3	->	2 (Iris Versicolor)
y:	5.1	3.8	1.9	0.4		

Table 4. Example from the *Iris* database.

As an example, consider two training instances A and B as shown in Table 4, and a new input vector $\boldsymbol{y}$ to be classified. For attribute a=1, the discretized values for A,

B, and $\boldsymbol{y}$ are 1, 2, and 2, respectively. Using values from Figure 6, the distance for attribute 1 between $\boldsymbol{y}$ and A is

$$|.867-.485|^2 + |.1-.455|^2 + |.033-.061|^2 = .273$$

while the distance between $\boldsymbol{y}$ and B is 0, since they have the same discretized value.

Note that $\boldsymbol{y}$ and B have values on different ends of range 2, and are not actually nearly as close as $\boldsymbol{y}$ and A are. In spite of this fact, the discretized distance function says that $\boldsymbol{y}$ and B are equal because they happen to fall into the same discretized range.

IVDM uses interpolation to alleviate such problems. IVDM assumes that the $P_{a,x,c}$ values hold true only at the midpoint of each range, and interpolates between midpoints to find P for other attribute values.

Figure 7 shows the P values for the second output class (*Iris Versicolor*) as a function of the first attribute value (*sepal length*). The dashed line indicates what P value is used by DVDM, and the solid line shows what IVDM uses.

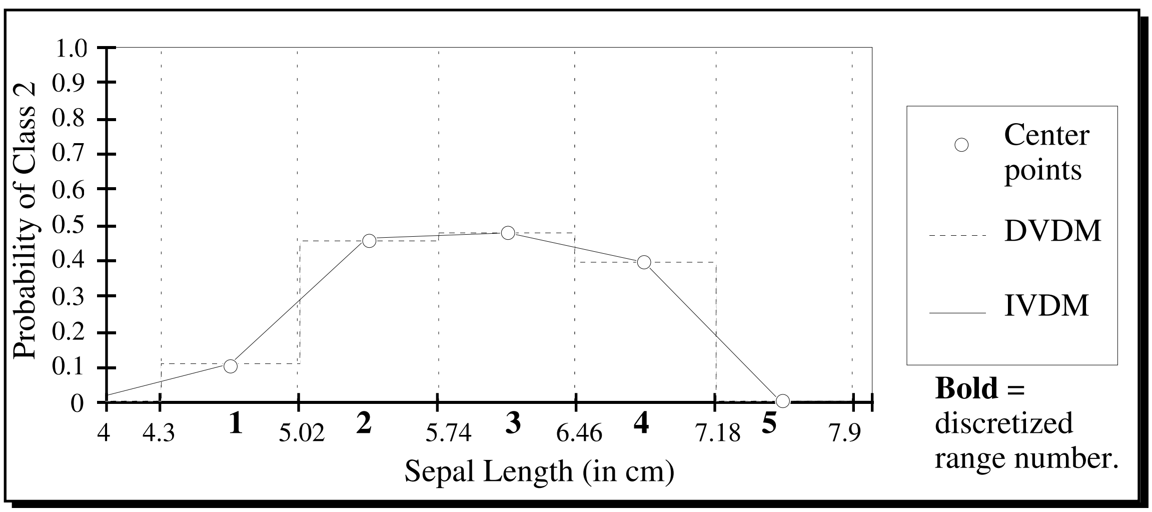


Figure 7. $P_{1,x,2}$ values from the DVDM and IVDM for attribute 1, class 2 of the *Iris* database.

The distance function for the Interpolated Value Difference Metric is defined as

$$IVDM(\boldsymbol{x},\boldsymbol{y}) = \sum_{a=1}^{m} ivdm_a(x_a, y_a)^2 \tag{21}$$

where $ivdm_a$ is defined as

$$ivdm_a(x,y) = \begin{cases} vdm_a(x,y) & \text{if } a \text{ is discrete} \\ \sum_{c=1}^{C} \left| p_{a,c}(x) - p_{a,c}(y) \right|^2 & \text{otherwise} \end{cases} \tag{22}$$

The formula for determining the interpolated probability value $p_{a,c}(x)$ of a continuous value x for attribute a and class c is

$$p_{a,c}(x) = P_{a,u,c} + \left(\frac{x - mid_{a,u}}{mid_{a,u+1} - mid_{a,u}} \right) * (P_{a,u+1,c} - P_{a,u,c}) \quad (23)$$

In this equation, $mid_{a,u}$ and $mid_{a,u+1}$ are midpoints of two consecutive discretized ranges such that $mid_{a,u} \leq x < mid_{a,u+1}$. $P_{a,u,c}$ is the probability value of the discretized range u, which is taken to be the probability value of the midpoint of range u (and similarly for $P_{a,u+1,c}$). The value of u is found by first setting $u = discretize_a(x)$, and then subtracting 1 from u if $x < mid_{a,u}$. The value of $mid_{a,u}$ can then be found as follows

$$mid_{a,u} = min_a + width_a * (u+.5) \quad (24)$$

Figure 8 shows the values of $p_{a,c}(x)$ for attribute a=1 of the *Iris* database for all three output classes (i.e. c=1, 2, and 3). Since there are no data points outside the range $min_a..max_a$, the probability value $P_{a,u,c}$ is taken to be 0 when $u < 1$ or $u > s$, which can be seen visually by the diagonal lines sloping toward zero on the outer edges of the graph. Note that the sum of the probabilities for the three output classes sum to 1.0 at every point from the midpoint of range 1 through the midpoint of range 5.

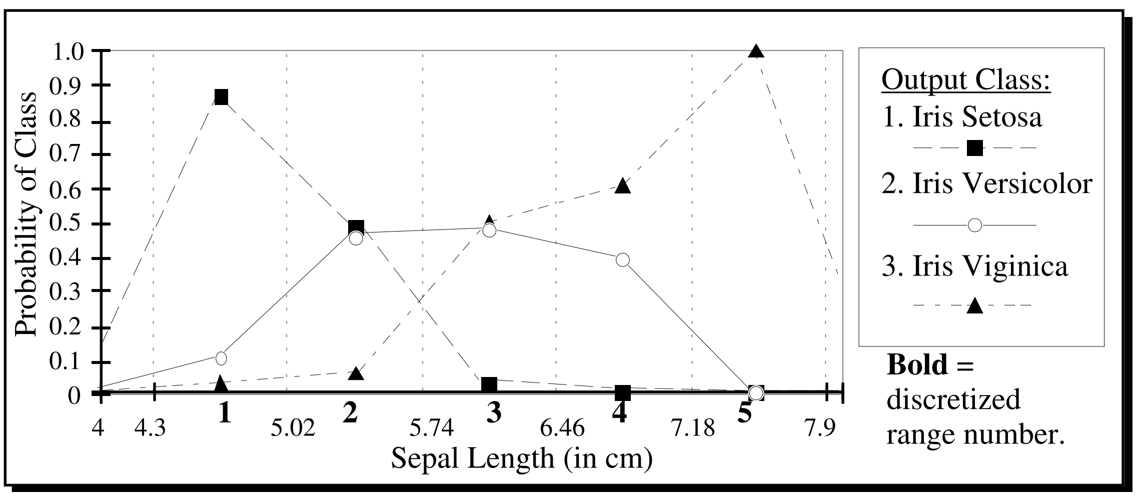


Figure 8. Interpolated probability values for attribute 1 of the *Iris* database.

Using IVDM on the example instances in Table 4, the values for the first attribute are not discretized as they are with DVDM, but are used to find interpolated probability values. In that example, ***y*** has a value of 5.1, so $p_{1,c}(x)$ interpolates between midpoints 1 and 2, returning the values shown in Table 5 for each of the three classes. Instance *A* has a value of 5.0, which also falls between midpoints 1 and 2, but instance *B* has a value of 5.7, which falls between midpoints 2 and 3.

As can be seen from Table 5, IVDM (using the single-attribute distance function *ivdm*) returns a distance which indicates that ***y*** is closer to *A* than *B* (for the first attribute), which is certainly the case here. DVDM (using the discretized *vdm*), on the other hand, returns a distance which indicates that the value of ***y*** is *equal* to that of *B*, and quite far from *A*, illustrating the problems involved with using discretization.

	value	$p_{1,1}(v)$	$p_{1,2}(v)$	$p_{1,3}(v)$	$ivdm_1(v,y)$	$vdm_1(v,y)$
A	5.0	.687	.268	.046	**.005**	.273
B	5.7	.281	.463	.256	**.188**	0
y	5.1	.634	.317	.050		

Table 5. Example of *ivdm* vs. *vdm.*

The IVDM and DVDM algorithms were implemented and tested on 48 datasets from the UCI machine learning databases. The results for the 34 datasets that contain at least some continuous attributes are shown in Table 6. (Since IVDM and DVDM are equivalent on domains with only discrete attributes, the results on the remaining

Database	DVDM	IVDM
Annealing	94.99	**96.11** *
Australian	**83.04** *	80.58
Bridges	56.73	**60.55**
Credit Screening	**80.14**	**80.14**
Echocardiogram	**100.00**	**100.00**
Flag	**58.76**	57.66
Glass	56.06	**70.54** *
Heart Disease	80.37	**81.85**
Heart (Cleveland)	**79.86**	78.90
Heart (Hungarian)	**81.30**	80.98
Heart (Long-Beach-Va)	**71.00**	66.00
Heart (More)	72.29	**73.33**
Heart (Swiss)	**88.59**	87.88
Hepatitis	80.58	**82.58**
Horse-Colic	76.75	**76.78**
Image Segmentation	92.38	**92.86**
Ionosphere	**92.60**	91.17
Iris	92.00	**94.67**
Liver Disorders	55.04	**58.23**
Pima Indians Diabetes	**71.89**	69.28
Satellite Image	87.06	**89.79** *
Shuttle	96.17	**99.77** *
Sonar	78.45	**84.17**
Thyroid (Allbp)	94.86	**95.32**
Thyroid (Allhyper)	96.93	**97.86** *
Thyroid (Allhypo)	89.36	**96.07** *
Thyroid (Allrep)	96.86	**98.43** *
Thyroid (Dis)	**98.29**	98.04
Thyroid (Hypothyroid)	93.01	**98.07** *
Thyroid (Sick)	88.24	**95.07** *
Thyroid (Sick-Euthyroid)	88.82	**96.86** *
Vehicle	63.72	**69.27** *
Vowel	91.47	**97.53** *
Wine	94.38	**97.78** *
Average:	83.08	**85.22**

Table 6. Generalization for DVDM vs. IVDM.

datasets are deferred to Section 6.) 10-fold cross-validation was again used, and the average accuracy for each database over all 10 trials is shown in Table 6. Bold values indicate which value was highest for each dataset. Asterisks (*) indicates that the difference is statistically significant at a 90% confidence level or higher, using a two-tailed paired *t*-test.

On this set of datasets, IVDM had a higher average generalization accuracy overall than the discretized algorithm. IVDM obtained higher generalization accuracy than DVDM in 23 out of 34 cases, 13 of which were significant at the 90% level or above. DVDM had a higher accuracy in 9 cases, but only one of those had a difference that was statistically significant.

These results indicate that the interpolated distance function is typically more appropriate than the discretized value difference metric for applications with one or more continuous attributes. Section 6 contains further comparisons of IVDM with other distance functions.

5. Windowed Value Difference Metric (WVDM)

The IVDM algorithm can be thought of as sampling the value of $P_{a,u,c}$ at the midpoint $mid_{a,u}$ of each discretized range u. P is sampled by first finding the instances that have a value for attribute a in the range $mid_{a,u} \pm w_a / 2$. $N_{a,u}$ is incremented once for each such instance, and $N_{a,u,c}$ is also incremented for each instance whose output class is c, after which $P_{a,u,c} = N_{a,u,c} / N_{a,u}$ is computed. IVDM then interpolates between these sampled points to provide a continuous but rough approximation to the function $p_{a,c}(x)$. It is possible to sample P at more points and thus provide a closer approximation to the function $p_{a,c}(x)$, which may in turn provide for more accurate distance measurements between values.

Figure 9 shows pseudo-code for the Windowed Value Difference Metric (WVDM). The WVDM samples the value of $P_{a,x,c}$ at each value x occurring in the training set for each attribute a, instead of only at the midpoints of each range. In fact, the discretized ranges are not even used by WVDM on continuous attributes, except to determine an appropriate *window width*, w_a, which is the same as the range width used in DVDM and IVDM. The pseudo-code for the learning algorithm used to determine $P_{a,x,c}$ for each attribute value x is given in Figure 9.

For each value x occurring in the training set for attribute a, P is sampled by finding the instances that have a value for attribute a in the range $x \pm w_a / 2$, and then computing $N_{a,x}$, $N_{a,x,c}$, and $P_{a,x,c} = N_{a,x,c} / N_{a,x}$ as before. Thus, instead of having a fixed number s of sampling points, a *window* of instances, centered on each training instance, is used for determining the probability at a given point. This technique is similar in concept to *shifted histogram estimators* [Rosenblatt, 1956] and to *Parzen window* techniques [Parzen, 1962].

For each attribute the values are sorted (using an O($n$log$n$) sorting algorithm) so as to allow a sliding window to be used and thus collect the needed statistics in O(n) time for each attribute. The sorted order is retained for each attribute so that a binary search can be performed in O(log n) time during generalization.

Values occurring between the sampled points are interpolated just as in IVDM, except that there are now many more points available, so a new value will be interpolated between two closer, more precise values than with IVDM.

```
Define:
    instance[a][1..n] as the list of all n instances in T sorted in ascending order by attribute a.
    instance[a][i].val[a]  as the value of attribute a for instance[a][i].
    x                  as the center value of the current window, i.e., x=instance[a][i].val[a].
    p[a][i][c]         as the probability P_{a,x,c} that the output class is c given the input value x for
                           attribute a.  Note that i is an index, the not value itself.
    N[c]               as the number N_{a,x,c} of instances in the current window with output class c.
    N                  as the total number N_{a,x} of instances in the current window.
    instance[a][in]    as the first instance in the window.
    instance[a][out]   as the first instance outside the window.  (i.e., the window contains
                           instances instance[a][in..out-1]).
    w[a]               as the window width for attribute a.

LearnWVDM(training set T)
    For each continuous attribute a
        Sort instance[a][1..n] in ascending order by attribute a, using a quicksort.
        Initialize N and N[c] to 0, and in and out to 1 (i.e., start with an empty window).
        For each i=1..n
            Let x=instance[a][i].val[a].
            // Expand window to include all instances in range
            While (out < n) and (instance[a][out].val[a] < (x + w[a]/2))
                Increment N[c], where c=the class of instance[a][out].
                Increment N.
                Increment out.
            // Shrink window to exclude instances no longer in range
            While (in < out) and (instance[a][in].val[a] < (x - w[a]/2))
                Decrement N[c], where c=the class of instance[a][in].
                Decrement N.
                Increment in.
            // Compute the probability value for each class from the current window
            for each class c=1..C
                p[a][i][c] = N[c] / N.  (i.e., P_{a,x,c} = N_{a,x,c} / N_{a,x}).
    Return the 3-D array p[a][i][c].
```

Figure 9. Pseudo code for the WVDM learning algorithm.

The pseudo-code for the interpolation algorithm is given in Figure 10. This algorithm takes a value x for attribute a and returns a vector of C probability values $P_{a,x,c}$ for c=1..C. It first does a binary search to find the two consecutive instances in

```
WVDM_Find_P(attribute a,continuous value x)
    // Find P_{a,x,c} for c=1..C, given a value x for attribute a.
    Find i such that instance[a][i].val[a] ≤ x ≤ instance[a][i+1].val[a] (binary search).
    x1 = instance[a][i].val[a]       (unless i<1, in which case x1=min[a] - (w[a] / 2))
    x2 = instance[a][i+1].val[a]     (unless i>n, in which case x2=max[a] + (w[a] / 2))
    For each class c=1..C
        p1=p[a][i][c]        (unless i<1, in which case p1=0)
        p2=p[a][i+1][c]      (unless i>n, in which case p2=0)
        P_{a,x,c} = p1 + ((x-x1)/(x2-x1)) * (p2 - p1)
    Return array P_{a,x,1..C}.
```

Figure 10. Pseudo-code for the WVDM probability interpolation (see Figure 9 for definitions).

the sorted list of instances for attribute a that surround x. The probability for each class is then interpolated between that stored for each of these two surrounding instances. (The exceptions noted in parenthesis handle outlying values by interpolating towards 0 as is done in IVDM.)

Once the probability values for each of an input vector's attribute values are computed, they can be used in the *vdm* function just as the discrete probability values are.

The WVDM distance function is defined as

$$WVDM(\boldsymbol{x},\boldsymbol{y}) = \sum_{a=1}^{m} wvdm_a(x_a, y_a)^2 \tag{25}$$

and $wvdm_a$ is defined as

$$wvdm_a(x,y) = \begin{cases} vdm_a(x,y) & \text{if } a \text{ is discrete} \\ \sum_{c=1}^{C} \left| P_{a,x,c} - P_{a,y,c} \right|^2 & \text{otherwise} \end{cases} \tag{26}$$

where $P_{a,x,c}$ is the interpolated probability value for the continuous value x as computed in Figure 10. Note that we are typically finding the distance between a new input vector and an instance in the training set. Since the instances in the training set were used to define the probability at each of their attribute values, the binary search and interpolation is unnecessary for training instances because they can immediately recall their stored probability values, unless pruning techniques have been used.

One drawback to this approach is the increased storage needed to retain C probability values for each attribute value in the training set. Execution time is not significantly increased over IVDM or DVDM. (See Section 6.2 for a discussion on efficiency considerations).

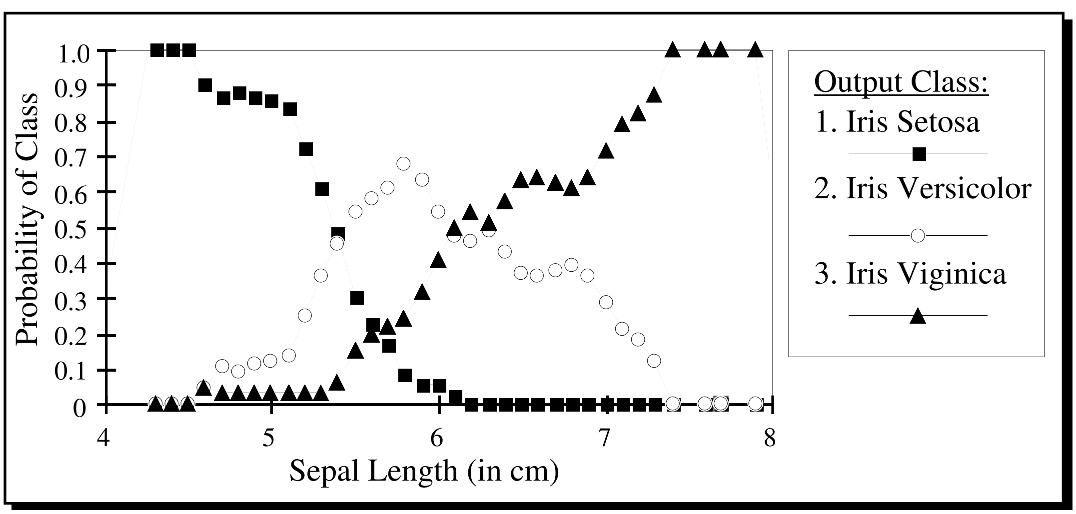


Figure 11. Example of the WVDM probability landscape.

Figure 11 shows the probability values for each of the three classes for the first attribute of the *Iris* database again, this time using the windowed sampling technique. Comparing Figure 11 with Figure 8 reveals that on this attribute IVDM provides approximately the same overall shape, but misses much of the detail. For example, the peak occurring for output class 2 at approximately *sepal length*=5.75. In Figure 8 there is a flat line which misses this peak entirely, due mostly to the somewhat arbitrary position of the midpoints at which the probability values are sampled.

Table 7 summarizes the results of testing the WVDM algorithm on the same datasets as DVDM and IVDM. A bold entry again indicates the highest of the two accuracy measurements, and an asterisk (*) indicates a difference that is statistically significant at the 90% confidence level, using a two-tailed paired *t*-test.

Database	DVDM	WVDM
Annealing	94.99	**95.87**
Australian	**83.04**	82.46
Bridges	**56.73**	56.64
Credit Screening	80.14	**81.45**
Echocardiogram	**100.00**	98.57
Flag	**58.76**	58.74
Glass	56.06	**71.49** *
Heart Disease	80.37	**82.96**
Heart (Cleveland)	79.86	**80.23**
Heart (Hungarian)	**81.30**	79.26
Heart (Long-Beach-Va)	**71.00**	68.00
Heart (More)	72.29	**73.33**
Heart (Swiss)	88.59	**88.72**
Hepatitis	**80.58**	79.88
Horse-Colic	**76.75**	74.77
Image Segmentation	92.38	**93.33**
Ionosphere	**92.60**	91.44
Iris	92.00	**96.00**
Liver Disorders	55.04	**57.09**
Pima Indians Diabetes	**71.89**	70.32
Satellite Image	87.06	**89.33** *
Shuttle	96.17	**99.61** *
Sonar	78.45	**84.19**
Thyroid (Allbp)	94.86	**95.29**
Thyroid (Allhyper)	96.93	**97.50**
Thyroid (Allhypo)	89.36	**90.18**
Thyroid (Allrep)	96.86	**97.07**
Thyroid (Dis)	**98.29**	98.00
Thyroid (Hypothyroid)	93.01	**96.96** *
Thyroid (Sick)	88.24	**97.11** *
Thyroid (Sick-Euthyroid)	88.82	**94.40** *
Vehicle	63.72	**65.37** *
Vowel	91.47	**96.21** *
Wine	94.38	**97.22** *
Average:	83.08	**84.68**

Table 7. Generalization of WVDM vs. DVDM.

On this set of databases, WVDM was an average of 1.6% more accurate than DVDM overall. WVDM had a higher average accuracy than DVDM on 23 out of the 34 databases, and was significantly higher on 9, while DVDM was only higher on 11 databases, and none of those differences were statistically significant.

Section 6 provides further comparisons of WVDM with other distance functions, including IVDM.

6. Empirical Comparisons and Analysis of Distance Functions

This section compares the distance functions discussed in this paper. A nearest neighbor classifier was implemented using each of six different distance functions: Euclidean (normalized by standard deviation) and HOEM as discussed in Section 2; HVDM as discussed in Section 3; DVDM and IVDM as discussed in Section 4; and WVDM as discussed in Section 5. Figure 12 summarizes the definition of each distance function.

All functions use the same overall distance function: $D(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{a=1}^{m} d_a(x_a,y_a)^2}$

Distance Function	Definition of $d_a(x_a,y_a)$ for each attribute type: Continuous	Linear Discrete	Nominal
Euclidean	$\frac{x_a - y_a}{\sigma_a}$	←	$\frac{x_a - y_a}{\sigma_a}$
HOEM	$\frac{x_a - y_a}{range_a}$	←	0 if $x_a = y_a$ 1 if $x_a \neq y_a$
HVDM	$\frac{x_a - y_a}{4\sigma_a}$	←	$\sqrt{vdm_a(x_a,y_a)}$
DVDM	$vdm_a(disc_a(x_a),disc_a(y_a))$	→	$\sqrt{vdm_a(x_a,y_a)}$
IVDM	$ivdm_a(x_a,y_a)$ Interpolate probabilities from range midpoints.	→	$\sqrt{vdm_a(x_a,y_a)}$
WVDM	$wvdm_a(x_a,y_a)$ Interpolate probabilities from adjacent values.	→	$\sqrt{vdm_a(x_a,y_a)}$

where $range_a = max_a - min_a$, and $vdm_a(x,y) = \sum_{c=1}^{C} \left|P_{a,x,c} - P_{a,y,c}\right|^2$

Figure 12. Summary of distance function definitions.

Each distance function was tested on 48 datasets from the UCI machine learning databases, again using 10-fold cross-validation. The average accuracy over all 10 trials is reported for each test in Table 8. The highest accuracy achieved for each dataset is shown in bold. The names of the three new distance functions presented in this paper (HVDM, IVDM and WVDM) are also shown in bold to identify them.

	Distance Function							# of inputs		
Database	Euclid	HOEM	**HVDM**	DVDM	**IVDM**	**WVDM**	#Inst.	Con	Int	Nom
Annealing	94.99	94.61	94.61	94.99	**96.11**	95.87	798	6	3	29
Audiology	60.50	72.00	**77.50**	**77.50**	**77.50**	**77.50**	200	0	0	69
Audiology (test)	41.67	75.00	**78.33**	**78.33**	**78.33**	**78.33**	26	0	0	69
Australian	80.58	81.16	81.45	**83.04**	80.58	82.46	690	6	0	8
Breast Cancer	94.99	95.28	94.99	**95.57**	**95.57**	**95.57**	699	0	9	0
Bridges	58.64	53.73	59.64	56.73	**60.55**	56.64	108	1	3	7
Credit Screening	78.99	81.01	80.87	80.14	80.14	**81.45**	690	6	0	9
Echocardiogram	94.82	94.82	94.82	**100.00**	**100.00**	98.57	132	7	0	2
Flag	48.95	48.84	55.82	**58.76**	57.66	58.74	194	3	7	18
Glass	**72.36**	70.52	**72.36**	56.06	70.54	71.49	214	9	0	0
Heart Disease	72.22	75.56	78.52	80.37	81.85	**82.96**	270	5	2	6
Heart (Cleveland)	73.94	74.96	76.56	79.86	78.90	**80.23**	303	5	2	6
Heart (Hungarian)	73.45	74.47	76.85	**81.30**	80.98	79.26	294	5	2	6
Heart (Long-Beach-Va)	**71.50**	71.00	65.50	71.00	66.00	68.00	200	5	2	6
Heart (More)	72.09	71.90	72.09	72.29	**73.33**	**73.33**	1541	5	2	6
Heart (Swiss)	**93.53**	91.86	89.49	88.59	87.88	88.72	123	5	2	6
Hepatitis	77.50	77.50	76.67	80.58	**82.58**	79.88	155	6	0	13
Horse-Colic	65.77	60.82	60.53	76.75	**76.78**	74.77	301	7	0	16
House-Votes-84	93.12	93.12	**95.17**	**95.17**	**95.17**	**95.17**	435	0	0	16
Image Segmentation	92.86	**93.57**	92.86	92.38	92.86	93.33	420	18	0	1
Ionosphere	86.32	86.33	86.32	**92.60**	91.17	91.44	351	34	0	0
Iris	94.67	95.33	94.67	92.00	94.67	**96.00**	150	4	0	0
LED+17 noise	42.90	42.90	**60.70**	**60.70**	**60.70**	**60.70**	10000	0	0	24
LED	**57.20**	**57.20**	56.40	56.40	56.40	56.40	1000	0	0	7
Liver Disorders	62.92	**63.47**	62.92	55.04	58.23	57.09	345	6	0	0
Monks-1	**77.08**	69.43	68.09	68.09	68.09	68.09	432	0	0	6
Monks-2	59.04	54.65	**97.50**	**97.50**	**97.50**	**97.50**	432	0	0	6
Monks-3	87.26	78.49	**100.00**	**100.00**	**100.00**	**100.00**	432	0	0	6
Mushroom	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	8124	0	1	21
Pima Indians Diabetes	71.09	70.31	71.09	**71.89**	69.28	70.32	768	8	0	0
Promoters	73.73	82.09	**92.36**	**92.36**	**92.36**	**92.36**	106	0	0	57
Satellite Image	90.21	**90.24**	90.21	87.06	89.79	89.33	4435	36	0	0
Shuttle	**99.78**	**99.78**	**99.78**	96.17	99.77	99.61	9253	9	0	0
Sonar	**87.02**	86.60	**87.02**	78.45	84.17	84.19	208	60	0	0
Soybean (Large)	87.26	89.20	90.88	**92.18**	**92.18**	**92.18**	307	0	6	29
Soybean (Small)	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	47	0	6	29
Thyroid (Allbp)	94.89	94.89	95.00	94.86	**95.32**	95.29	2800	6	0	22
Thyroid (Allhyper)	97.00	97.00	96.86	96.93	**97.86**	97.50	2800	6	0	22
Thyroid (Allhypo)	90.39	90.39	90.29	89.36	**96.07**	90.18	2800	6	0	22
Thyroid (Allrep)	96.14	96.14	96.11	96.86	**98.43**	97.07	2800	6	0	22
Thyroid (Dis)	98.21	98.21	98.21	**98.29**	98.04	98.00	2800	6	0	22
Thyroid (Hypothyroid)	93.42	93.42	93.36	93.01	**98.07**	96.96	3163	7	0	18
Thyroid (Sick-Euthyroid)	68.23	68.23	68.23	88.24	**95.07**	94.40	3163	7	0	18
Thyroid (Sick)	86.93	86.89	86.61	88.82	96.86	**97.11**	2800	6	0	22
Vehicle	**70.93**	70.22	70.93	63.72	69.27	65.37	846	18	0	0
Vowel	**99.24**	98.86	**99.24**	91.47	97.53	96.21	528	10	0	0
Wine	95.46	95.46	95.46	94.38	**97.78**	97.22	178	13	0	0
Zoo	97.78	94.44	**98.89**	**98.89**	**98.89**	**98.89**	90	0	0	16
Average:	80.78	81.29	83.79	84.06	**85.56**	85.24				

Table 8. Summary of Generalization Accuracy

Table 8 also lists the number of instances in each database ("#Inst."), and the number of continuous ("Con"), integer ("Int", i.e., linear discrete), and nominal ("Nom") input attributes.

On this set of 48 datasets, the three new distance functions (HVDM, IVDM and WVDM) did substantially better than Euclidean distance or HOEM. IVDM had the highest average accuracy (85.56%) and was almost 5% higher on average than Euclidean distance (80.78%), indicating that it is a more robust distance function on these datasets, especially those with nominal attributes. WVDM was only slightly lower than IVDM with 85.24% accuracy. Somewhat surprisingly, DVDM was slightly higher than HVDM on these datasets, even though it uses discretization instead of a linear distance on continuous attributes. All four of the VDM-based distance functions outperformed Euclidean distance and HOEM.

Out of the 48 datasets, Euclidean distance had the highest accuracy 11 times; HOEM was highest 7 times; HVDM, 14; DVDM, 19; IVDM, 25 and WVDM, 18.

For datasets with no continuous attributes, all four of the VDM-based distance functions (HVDM, DVDM, IVDM and WVDM) are equivalent. On such datasets, the VDM-based distance functions achieve an average accuracy of 86.6% compared to 78.8% for HOEM and 76.6% for Euclidean, indicating a substantial superiority on such problems.

For datasets with no nominal attributes, Euclidean and HVDM are equivalent, and all the distance functions perform about the same on average except for DVDM, which averages about 4% less than the others, indicating the detrimental effects of discretization. Euclidean and HOEM have similar definitions for applications without any nominal attributes, except that Euclidean is normalized by standard deviation while HOEM is normalized by the range of each attribute. It is interesting that the average accuracy over these datasets is slightly higher for Euclidean than HOEM, indicating that the standard deviation may provide better normalization on these datasets. However, the difference is small (less than 1%), and these datasets do not contain many outliers, so the difference is probably negligible in this case.

One disadvantage with scaling attributes by the standard deviation is that attributes which almost always have the same value (e.g., a boolean attribute that is almost always 0) will be given a large weight—not due to scale, but because of the relative frequencies of the attribute values. A related problem can occur in HVDM. If there is a very skewed class distribution (i.e., there are many more instances of some classes than others), then the P values will be quite small for some classes and quite large for others, and in either case the difference $|P_{a,x,c} - P_{a,y,c}|$ will be correspondingly small, and thus nominal attributes will get very little weight when compared to linear attributes. This phenomenon was noted by Ting [1994, 1996], where he recognized such problems on the *hypothyroid* dataset. Future research will address these normalization problems and look for automated solutions. Fortunately, DVDM, IVDM and WVDM do not suffer from either problem, because all attributes are scaled by the same amount in such cases, which may in part account for their success over HVDM in the above experiments. [See the appendix to this chapter for recent developments on handling skewed class distributions in HVDM.]

For datasets with both nominal and continuous attributes, HVDM is slightly higher than Euclidean distance on these datasets, which is in turn slightly higher than HOEM, indicating that the overlap metric may not be much of an improvement on heterogeneous databases. DVDM, IVDM and WVDM are all higher than Euclidean distance on such datasets, with IVDM again in the lead.

6.1. Effects of Sparse Data

Distance functions that use VDM require some statistics to determine distance. We therefore hypothesized that generalization accuracy might be lower for VDM-based distance functions than for Euclidean distance or HOEM when there was very little data available, and that VDM-based functions would increase in accuracy more slowly than the others as more instances were made available, until a sufficient number of instances allowed a reasonable sample size to determine good probability values.

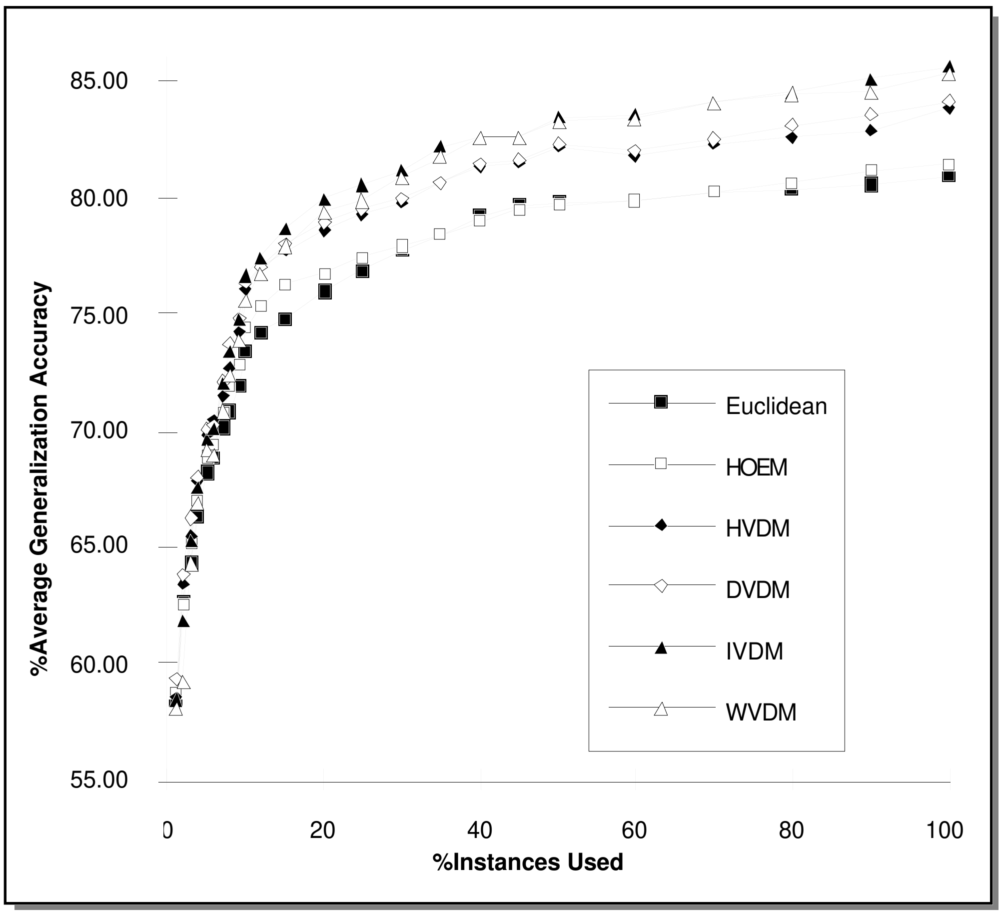


Figure 13. Average accuracy as the amount of data increases.

To test this hypothesis, the experiments used to obtain the results shown in Table 8 were repeated using only part of the available training data. Figure 13 shows how the generalization accuracy on the test set improves as the percentage of available training instances used for learning and generalization is increased from 1% to 100%. The generalization accuracy values shown are the averages over all 48 of the datasets in Table 8.

Surprisingly, the VDM-based distance functions increased in accuracy as fast or faster than Euclidean and HOEM even when there was very little data available. It may be that when there is very little data available, the random positioning of the sample data in the input space has a greater detrimental affect on accuracy than does the error in statistical sampling for VDM-based functions.

It is interesting to note from Figure 13 that the six distance functions seem to pair up into three distinct pairs. The interpolated VDM-based distance functions (IVDM and WVDM) maintain the highest accuracy, the other two VDM-based functions are next, and the functions based only on linear and overlap distance remain lowest from very early in the graph.

6.2. Efficiency Considerations

This section considers the storage requirements, learning speed, and generalization speed of each of the algorithms presented in this paper.

6.2.1. Storage

All of the above distance functions must store the entire training set, requiring O(nm) storage, where n is the number of instances in the training set and m is the number of input attributes in the application, unless some instance pruning technique is used. For the Euclidean and HOEM functions, this is all that is necessary, but even this amount of storage can be restrictive as n grows large.

For HVDM, DVDM, and IVDM, the probabilities $P_{a,x,c}$ for all m attributes (only discrete attributes for HVDM) must be stored, requiring O(mvC) storage, where v is the average number of attribute values for the discrete (or discretized) attributes and C is the number of output classes in the application. It is possible to instead store an array $D_{a,x,y} = vdm_a(x,y)$ for HVDM and DVDM, but the storage would be O(mv^2), which is only a savings when $C < v$.

For WVDM, C probability values must be stored for each continuous attribute value, resulting in O(nmC) storage which is typically much larger than O(mvC) because n is usually much larger than v (and cannot be less). It is also necessary to store a list of (pointers to) instances for each attribute, requiring an additional O(mn) storage. Thus the total storage for WVDM is O((C+2)nm) = O(Cnm).

Distance Function	Storage	Learning Time	Generalization Time
Euclidean	O(mn)	O(mn)	O(mn)
HOEM	O(mn)	O(mn)	O(mn)
HVDM	O(mn+mvC)	O(mn+mvC)	O(mnC) or O(mn)
DVDM	O(mn+mvC)	O(mn+mvC)	O(mnC) or O(mn)
IVDM	O(mn+mvC)	O(mn+mvC)	O(mnC) or O(mn)
WVDM	O(Cmn)	O($mn\log n$+mvC)	O(mnC)

Table 9. Summary of efficiency for six distance metrics.

Table 9 summarizes the storage requirements of each system. WVDM is the only one of these distance functions that requires significantly more storage than the others. For most applications, n is the critical factor, and all of these distance functions could be used in conjunction with instance pruning techniques to reduce storage requirements. See Section 7 for a list of several techniques to reduce the number of instances retained in the training set for subsequent generalization.

6.2.2. LEARNING SPEED

It takes nm time to read in a training set. It takes an additional $2nm$ time to find the standard deviation of the attributes for Euclidean distance, or just nm time to find the ranges for HOEM.

Computing VDM statistics for HVDM, DVDM and IVDM takes $mn+mvC$ time, which is approximately $O(mn)$. Computing WVDM statistics takes $mn\log n+mnC$ time, which is approximately $O(mn\log n)$.

In general, the learning time is quite acceptable for all of these distance functions.

6.2.3. GENERALIZATION SPEED

Assuming that each distance function must compare a new input vector to all training instances, Euclidean and HOEM take $O(mn)$ time. HVDM, IVDM and DVDM take $O(mnC)$ (unless $D_{a,x,y}$ has been stored instead of $P_{a,x,c}$ for HVDM, in which case the search is done in $O(mn)$ time). WVDM takes $O(\log n+mnC) = O(mnC)$ time.

Though m and C are typically fairly small, the generalization process can require a significant amount of time and/or computational resources as n grows large. Techniques such as k-d trees [Deng & Moore, 1995; Wess, Althoff & Derwand, 1994; Sproull, 1991] and *projection* [Papadimitriou & Bentley, 1980] can reduce the time required to locate nearest neighbors from the training set, though such algorithms may require modification to handle both continuous and nominal attributes. Pruning techniques used to reduce storage (as in Section 6.2.1) will also reduce the number of instances that must be searched for generalization.

7. Related Work

Distance functions are used in a variety of fields, including instance-based learning, neural networks, statistics, pattern recognition, and cognitive psychology (see Section 1 for references). Section 2 lists several commonly used distance functions involving numeric attributes.

Normalization is often desirable when using a linear distance function such as Euclidean distance so that some attributes do not arbitrarily get more weight than others. Dividing by the range or standard deviation to normalize numerical attributes is common practice. Turney [1993; Turney & Halasz, 1993] investigated *contextual normalization*, in which the standard deviation and mean used for normalization of continuous attributes depend on the context in which the input vector was obtained. In this paper we do not attempt to use contextual normalization, but instead use simpler methods of normalizing continuous attributes, and then focus on how to normalize appropriately between continuous and nominal attributes.

The Value Distance Metric (VDM) was introduced by Stanfill & Waltz [1986]. It uses attribute weights not used by the functions presented in this paper. The Modified Value Difference Metric (MVDM) [Cost & Salzberg, 1993; Rachlin et al., 1994] does not use attribute weights but instead uses instance weights. It is assumed that these systems use discretization [Lebowitz, 1985; Schlimmer, 1987] to handle continuous attributes.

Ventura [1995; Ventura & Martinez, 1995] explored a variety of discretization methods for use in systems that can use only discrete input attributes. He found that using discretization to preprocess data often degraded accuracy, and recommended that machine learning algorithms be designed to handle continuous attributes directly.

Ting [1994, 1996] used several different discretization techniques in conjunction with MVDM and IB1 [Aha, Kibler & Albert, 1991]. His results showed *improved* generalization accuracy when using discretization. Discretization allowed his algorithm to use MVDM on all attributes instead of using a linear distance on continuous attributes, and thus avoided some of the normalization problems discussed above in Sections 3.1 and 3.2. In this paper, similar results can be seen in the slightly higher results of DVDM (which also discretizes continuous attributes and then uses VDM) when compared to HVDM (which uses linear distance on continuous attributes). In this paper, DVDM uses equal-width intervals for discretization, while Ting's algorithms make use of more advanced discretization techniques.

Domingos [1995] uses a heterogeneous distance function similar to HVDM in his RISE system, a hybrid rule and instance-based learning system. However, RISE uses a normalization scheme similar to "N1" in Sections 3.1 and 3.2, and does not square individual attribute distances.

Mohri & Tanaka [1994] use a statistical technique called Quantification Method II (QM2) to derive attribute weights, and present distance functions that can handle both nominal and continuous attributes. They transform nominal attributes with m values into m boolean attributes, only one of which is on at a time, so that weights for each attribute can actually correspond to individual attribute values in the original data.

Turney [1994] addresses cross-validation error and voting (i.e. using values of $k > 1$) in instance-based learning systems, and explores issues related to selecting the parameter k (i.e., number of neighbors used to decide on classification). In this paper we use $k = 1$ in order to focus attention on the distance functions themselves, but accuracy would be improved on some applications by using $k > 1$.

IVDM and WVDM use nonparametric density estimation techniques [Tapia & Thompson, 1978] in determining values of P for use in computing distances. Parzen windows [Parzen, 1962] and shifting histograms [Rosenblatt, 1956] are similar in concept to these techniques, especially to WVDM. These techniques often use gaussian kernels or other more advanced techniques instead of a fixed-sized sliding window. We have experimented with gaussian-weighted kernels as well but results were slightly worse than either WVDM or IVDM, perhaps because of increased overfitting.

This paper applies each distance function to the problem of classification, in which an input vector is mapped into a discrete output class. These distance functions could also be used in systems that perform *regression* [Atkeson, Moore & Schaal, 1996; Atkeson, 1989; Cleveland & Loader, 1994], in which the output is a real value, often interpolated from nearby points, as in kernel regression [Deng & Moore, 1995].

As mentioned in Section 6.2 and elsewhere, pruning techniques can be used to reduce the storage requirements of instance-based systems and improve classification speed. Several techniques have been introduced, including IB3 [Aha, Kibler & Albert, 1991; Aha, 1992], the condensed nearest neighbor rule [Hart, 1968], the reduced nearest neighbor rule [Gates, 1972], the selective nearest neighbor rule [Ritter et al., 1975], typical instance based learning algorithm [Zhang, 1992], prototype methods [Chang, 1974], hyperrectangle techniques [Salzberg, 1991;

Wettschereck & Dietterich, 1995], rule-based techniques [Domingos, 1995], random mutation hill climbing [Skalak, 1994; Cameron-Jones, 1995] and others [Kibler & Aha, 1987; Tomek, 1976; Wilson, 1972].

8. Conclusions & Future Research Areas

There are many learning systems that depend on a reliable distance function to achieve accurate generalization. The Euclidean distance function and many other distance functions are inappropriate for nominal attributes, and the HOEM function throws away information and does not achieve much better accuracy than the Euclidean function itself.

The Value Difference Metric (VDM) was designed to provide an appropriate measure of distance between two nominal attribute values. However, current systems that use the VDM often discretize continuous data into discrete ranges, which causes a loss of information and often a corresponding loss in generalization accuracy.

This paper introduced three new distance functions. The Heterogeneous Value Difference Function (HVDM) uses Euclidean distance on linear attributes and VDM on nominal attributes, and uses appropriate normalization. The Interpolated Value Difference Metric (IVDM) and Windowed Value Difference Metric (WVDM) handle continuous attributes within the same paradigm as VDM. Both IVDM and WVDM provide classification accuracy which is higher on average than the discretized version of the algorithm (DVDM) on the datasets with continuous attributes that we examined, and they are both equivalent to DVDM on applications without any continuous attributes.

In our experiments on 48 datasets, IVDM and WVDM achieved higher average accuracy than HVDM, and also did better than DVDM, HOEM and Euclidean distance. IVDM was slightly more accurate than WVDM and requires less time and storage, and thus would seem to be the most desirable distance function on heterogeneous applications similar to those used in this paper. Properly normalized Euclidean distance achieves comparable generalization accuracy when there are no nominal attributes, so in such situations it is still an appropriate distance function.

The learning system used to obtain generalization accuracy results in this paper was a nearest neighbor classifier, but the HVDM, IVDM and WVDM distance functions can be used with a k-nearest neighbor classifier with $k > 1$ or incorporated into a wide variety of other systems to allow them to handle continuous values including instance-based learning algorithms (such as PEBLS), radial basis function networks, and other distance-based neural networks. These new distance metrics can also be used in such areas as statistics, cognitive psychology, pattern recognition and other areas where the distance between heterogeneous input vectors is of interest. These distance functions can also be used in conjunction with weighting schemes and other improvements that each system provides.

The new distance functions presented here show improved average generalization on the 48 datasets used in experimentation. It is hoped that these datasets are representative of the kinds of applications that we face in the real world, and that these new distance functions will continue to provide improved generalization accuracy in such cases.

Future research will look at determining under what conditions each distance function is appropriate for a particular application. We will also look closely at the problem at selecting the window width, and will look at the possibility of smoothing WVDM's probability landscape to avoid overfitting. The new distance functions will also be used in conjunction with a variety of weighting schemes to provide more robust generalization in the presence of noise and irrelevant attributes, as well as increase generalization accuracy on a wide variety of applications.

Appendix. Handling Skewed Class Distributions with HVDM.

[*Note: This appendix did not appear in the JAIR paper, but describes research done after its publication.*]

As mentioned in Sections 3 and 6, the HVDM distance metric is sensitive to skewed class distributions. If there are many more instances of some classes than others, then the probability $P_{a,x,c}$ that the class is c will on average be quite small (close to 0) for some classes and quite large (close to 1) for others, due to the *a priori* probability of the class. Whether the class is a common one or an uncommon one, the difference $|P_{a,x,c} - P_{a,y,c}|$ will be correspondingly small, and thus nominal attributes will get very little weight when compared to linear attributes. This phenomenon was noted by Ting [1994, 1996], where he recognized such problems on the *hypothyroid* dataset.

The following example illustrates this problem. Consider a single binary input attribute x in a problem with binary output classes. (There could be additional input attributes as well, but VDM would treat them separately). Table 10 shows the steps leading to the determination of the distance between the two values 0 and 1 for attribute x.

In this problem there are 110 instances, 10 of which are in class 0, and 100 of which are in class 1. Table 10(a) shows the number of instances in each class that have each input value for attribute x. Table 10(b) shows what the probability values are in each case, and Table 10(c) shows how these values are used to arrive at a difference of .1776 between values 0 and 1 of attribute x.

However, note that class 0 is much more highly correlated with x=1 than with x=0, while class 1 is much more highly correlated with x=0 than with x=1. In other words, the values 0 and 1 for attribute x are actually quite different, but this is not reflected by the attribute distance of .1776. Compare this to the attribute distance of 1.0 that would be assigned if the overlap distance were used.

$N_{a,x,c}$

x	*class* 0	*class* 1
0	1	80
1	9	20

$P_{a,x,c}$

x	*class* 0	*class* 1
0	1/81=.01234	80/81=.98765
1	9/29=.31034	20/29=.68966

$$vdm_a(0,1)=(.01234-.31034)^2 +(.98765-.68966)^2 =.0888+.08888 =\mathbf{.1776}$$

Table 10. Illustration of problems with HVDM in the presence of skewed class distributions. (a) Number of instances with each attribute value that occur in each class. (b) The probability of each class, given the attribute value. (c) The distance between x=0 and x=1 computed by vdm_a.

It is possible to replicate the instances in the less common class in order to get rid of the effects of skewed class distribution. In our example, we can accomplish this by including each instance of class 0 in the training set 10 times when computing VDM distances. The effect of doing this is shown in Table 11.

		class 0	*class* 1			*class* 0	*class* 1	
x	0	10	80	x	0	10/90=.1111	80/90=.8889	$vdm_a(0,1)=(.1111-.8181)^2$
	1	90	20		1	90/110=.8181	20/110=.1818	$+(.8889-.1818)^2$ $=.4999+..4999$ **=.9998**
		$N_{a,x,c}$				$P_{a,x,c}$		

Table 11. Effect of replicating instances so that class frequencies are equalized.

In this example, there are now 100 instances of each class, and as can be seen from Table 11, the distance between values 0 and 1 for attribute x is now almost 1, which is almost five times as large as before. This method brings out the fact that the values are indeed quite different for each class.

Replicating instances is not convenient, especially when the number of instances is the majority class is not a multiple of the number in the minority class. Fortunately, it is also not necessary. The $vdm_a(x,y)$ function can be modified such that it directly discounts the effect of the *a priori* probability of each class. This can be done by dividing $N_{a,x,c}$ by the total number of instances in each class. Let this modified value be $N'_{a,x,c}$. Then this value is defined as

$$N'_{a,x,c} = \frac{N_{a,x,c}}{\sum_{c=1}^{C} N_{a,x,c}} \tag{27}$$

where C is the number of output classes. The probability is computed with the new value $N'_{a,x,c}$ using the same functions as in (9),

$$P'_{a,x,c} = \frac{N'_{a,x,c}}{N'_{a,x}} \tag{28}$$

where $N'_{a,x}$ is the sum of $N'_{a,x,c}$ over all classes, i.e.,

$$N'_{a,x} = \sum_{c=1}^{C} N'_{a,x,c} \tag{29}$$

Applying this technique to the example above yields the probabilities shown in Table 12. As can be seen from Table 12(c), the final probabilities are identical to those obtained by replicating the instances in the minority class, which means that the distance between values 0 and 1 using this technique will be .9998 as in Table 11.

The modified version of the HVDM that uses this technique is called the *Normalized Heterogeneous Value Difference Metric* (NHVDM).

		class 0	class 1			class 0	class 1			class 0	class 1
x	0	1	80	x	0	1/10=.1	80/100=.8	x	0	.1/.9=.1111	.8/.9=.8889
	1	9	20		1	9/10=.9	20/100=.2		1	.9/1.1=.8181	.2/1.1=.1818
		$N_{a,x,c}$				$N'_{a,x,c}$				$P'_{a,x,c}$	

Table 12. Normalizing VDM to reduce the effect of skewed class distributions.

In experiments on the same datasets as those used in Section 6, the NHVDM had lower accuracy on average that HVDM (82.73% for NHVDM vs. 83.79% for HVDM). However, it did achieve slightly higher accuracy on the *hypothyroid* dataset (93.42% vs. 93.36%), which, as mentioned in Section 6, has a skewed class distribution and in fact led to the development of this extension. It is interesting to note that 95% of the instances in the *hypothyroid* dataset are of the same class, which means that almost 95% accuracy can be obtained simply by choosing the most common class. However, the amount of *information* added by the classification of these models may be much higher than that achieved by the simple approach, and thus are of value [Zarndt, 1995].

We recommend at this stage that the NHVDM distance function be used instead of HVDM when there are both nominal and continuous attributes and when the classes have very different frequencies. For problems with nearly equal class distributions, HVDM was better empirically in our experiments. IVDM actually did better than both of these alternatives in our experiments, and avoids the problem of skewed distributions by the fact that all attributes' distances will be reduced by a similar amount, thus nullifying the effect.

References

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.

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

Atkeson, Chris, (1989). Using local models to control movement. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2*. San Mateo, CA: Morgan Kaufmann.

Atkeson, Chris, Andrew Moore, and Stefan Schaal, (1996). Locally weighted learning. To appear in *Artificial Intelligence Review*.

Batchelor, Bruce G., (1978). *Pattern Recognition: Ideas in Practice*. New York: Plenum Press, pp. 71-72.

Biberman, Yoram, (1994). A Context Similarity Measure. *In Proceedings of the European Conference on Machine Learning (ECML-94)*. Catalina, Italy: Springer Verlag, pp. 49-63.

Broomhead, D. S., and D. Lowe (1988). Multi-variable functional interpolation and adaptive networks. *Complex Systems*, vol. 2, pp. 321-355.

Cameron-Jones, R. M., (1995). Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing. *In Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence*, pp. 99-106.

Carpenter, Gail A., and Stephen Grossberg, (1987). A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine. *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115.

Chang, Chin-Liang, (1974). Finding Prototypes for Nearest Neighbor Classifiers. *IEEE Transactions on Computers*, vol. 23, no. 11, pp. 1179-1184.

Cleveland, W. S., and C. Loader, (1994). Computational Methods for Local Regression. Technical Report 11, Murray Hill, NJ: AT&T Bell Laboratories, Statistics Department.

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

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, pp. 21-27.

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

Deng, Kan, and Andrew W. Moore, (1995). Multiresolution Instance-Based Learning. *To appear in The Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95).*

Diday, Edwin, (1974). Recent Progress in Distance and Similarity Measures in Pattern Recognition. *Second International Joint Conference on Pattern Recognition*, pp. 534-539.

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).*

Dudani, Sahibsingh A., (1976). The Distance-Weighted *k*-Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, no. 4, April 1976, pp. 325-327.

Gates, G. W., (1972). The Reduced Nearest Neighbor Rule. *IEEE Transactions on Information Theory*, vol. IT-18, no. 3, pp. 431-433.

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.

Hart, P. E., (1968). The Condensed Nearest Neighbor Rule. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, vol. 14, pp. 515-516.

Hecht-Nielsen, R., (1987). Counterpropagation Networks. *Applied Optics*, vol. 26, no. 23, pp. 4979-4984.

Kibler, D., and David W. Aha, (1987). Learning representative exemplars of concepts: An initial case study. *Proceedings of the Fourth International Workshop on Machine Learning*. Irvine, CA: Morgan Kaufmann, pp. 24-30.

Kohonen, Teuvo, (1990). The Self-Organizing Map. *In Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464-1480.

Lebowitz, Michael, (1985). Categorizing Numeric Information for Generalization. *Cognitive Science*, vol. 9, pp. 285-308.

Merz, C. J., and P. M. Murphy, (1996). *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: http://www.ics.uci.edu/~mlearn/MLRepository.html.

Michalski, Ryszard S., Robert E. Stepp, and Edwin Diday, (1981). A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts. *Progress in Pattern Recognition*, vol. 1, Laveen N. Kanal and Azriel Rosenfeld (Eds.). New York: North-Holland, pp. 33-56.

Mitchell, Tom M., (1980). The Need for Biases in Learning Generalizations. in J. W. Shavlik & T. G. Dietterich (Eds.), *Readings in Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1990, pp. 184-191.

Mohri, Takao, and Hidehiko Tanaka, (1994). "An Optimal Weighting Criterion of Case Indexing for both Numeric and Symbolic Attributes. In D. W. Aha (Ed.), *Case-Based Reasoning: Papers from the 1994 Workshop*, Technical Report WS-94-01. Menlo Park, CA: AIII Press, pp. 123-127.

Nadler, Morton, and Eric P. Smith, (1993). *Pattern Recognition Engineering*. New York: Wiley, pp. 293-294.

Nosofsky, Robert M., (1986). Attention, Similarity, and the Identification-Categorization Relationship. *Journal of Experimental Psychology: General*, vol. 115, no. 1, pp. 39-57.

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.

Parzen, Emanuel, (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*. vol. 33, pp. 1065-1076.

Quinlan, J. R., (1989). Unknown Attribute Values in Induction. *In Proceedings of the 6th International Workshop on Machine Learning*. San Mateo, CA: Morgan Kaufmann, pp. 164-168.

Rachlin, John, Simon Kasif, Steven Salzberg, David W. Aha, (1994). Towards a Better Understanding of Memory-Based and Bayesian Classifiers. *In Proceedings of the Eleventh International Machine Learning Conference.* New Brunswick, NJ: Morgan Kaufmann, pp. 242-250.

Renals, Steve, and Richard Rohwer, (1989). Phoneme Classification Experiments Using Radial Basis Functions. *In Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'89)*, vol. 1, pp. 461-467.

Ritter, G. L., H. B. Woodruff, S. R. Lowry, and T. L. Isenhour, (1975). An Algorithm for a Selective Nearest Neighbor Decision Rule. *IEEE Transactions on Information Theory*, vol. 21, no. 6, pp. 665-669.

Rosenblatt, Murray, (1956). Remarks on Some Nonparametric Estimates of a Density Function. *Annals of Mathematical Statistics*. vol. 27, pp. 832-835.

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

Salzberg, Steven, (1991). A Nearest Hyperrectangle Learning Method. *Machine Learning*, vol. 6, pp. 277-309.

Schaffer, Cullen, (1993). Selecting a Classification Method by Cross-Validation. *Machine Learning*, vol. 13, no. 1.

Schaffer, Cullen, (1994). A Conservation Law for Generalization Performance. *In Proceedings of the Eleventh International Conference on Machine Learning (ML'94)*, Morgan Kaufmann, 1994.

Schlimmer, Jeffrey C., (1987). Learning and Representation Change. *In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87)*, vol. 2, pp. 511-535.

Skalak, D. B., (1994). Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning (ML94).* Morgan Kaufmann, pp. 293-301.

Sproull, Robert F., (1991). Refinements to Nearest-Neighbor Searching in *k*-Dimensional Trees. *Algorithmica*, vol. 6, pp. 579-589.

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

Tapia, Richard A., and James R. Thompson, (1978). *Nonparametric Probability Density Estimation*. Baltimore, MD: The Johns Hopkins University Press.

Ting, Kai Ming, (1994). Discretization of Continuous-Valued Attributes and Instance-Based Learning. Technical Report no. 491, Basser Department of Computer Science, University of Sydney, Australia.

Ting, Kai Ming, (1996). Discretisation in Lazy Learning. To appear in the special issue on Lazy Learning in *Artificial Intelligence Review*.

Tomek, Ivan, (1976). An Experiment with the Edited Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 6, June 1976, pp. 448-452.

Turney, Peter, (1994). Theoretical Analyses of Cross-Validation Error and Voting in Instance-Based Learning. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, pp. 331-360.

Turney, Peter, (1993). Exploiting context when learning to classify. *In Proceedings of the European Conference on Machine Learning*. Vienna, Austria: Springer-Verlag, pp. 402-407.

Turney, Peter, and Michael Halasz, (1993). Contextual Normalization Applied to Aircraft Gas Turbine Engine Diagnosis. *Journal of Applied Intelligence*, vol. 3, pp. 109-129.

Tversky, Amos, (1977). Features of Similarity. *Psychological Review*, vol. 84, no. 4, pp. 327-352.

Ventura, Dan, (1995). *On Discretization as a Preprocessing Step for Supervised Learning Models*, Master's Thesis, Department of Computer Science, Brigham Young University.

Ventura, Dan, and Tony R. Martinez (1995). An Empirical Comparison of Discretization Methods. *In Proceedings of the Tenth International Symposium on Computer and Information Sciences*, pp. 443-450.

Wasserman, Philip D., (1993). *Advanced Methods in Neural Computing*. New York, NY: Van Nostrand Reinhold, pp. 147-176.

Wess, Stefan, Klaus-Dieter Althoff and Guido Derwand, (1994). Using *k*-d Trees to Improve the Retrieval Step in Case-Based Reasoning. Stefan Wess, Klaus-Dieter Althoff, & M. M. Richter (Eds.), *Topics in Case-Based Reasoning*. Berlin: Springer-Verlag, pp. 167-181.

Wettschereck, Dietrich, and Thomas G. Dietterich, (1995). An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms. *Machine Learning*, vol. 19, no. 1, pp. 5-28.

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.

Wilson, D. Randall, and Tony R. Martinez, (1993). The Potential of Prototype Styles of Generalization. In *Proceedings of the Sixth Australian Joint Conference on Artifical Intelligence (AI'93)*, pp. 356-361.

Wilson, D. Randall, and Tony R. Martinez, (1996). Heterogeneous Radial Basis Functions. In *Proceedings of the International Conference on Neural Networks (ICNN'96)*, vol. 2, pp. 1263-1267.

Wilson, Dennis L., (1972). Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, pp. 408-421.

Wolpert, David H., (1993). On Overfitting Avoidance as Bias. Technical Report SFI TR 92-03-5001. Santa Fe, NM: The Santa Fe Institute.

Zhang, Jianping, (1992). Selecting Typical Instances in Instance-Based Learning. *Proceedings of the Ninth International Conference on Machine Learning*.

Part III

Instance Set Reduction

"2B|~2B?" —William Shakespeare.

One of the main disadvantages of the basic nearest neighbor algorithm is its large storage requirements and slow classification speed. *Probabilistic neural networks* have very similar drawbacks and similar techniques can be used to address the weaknesses of both models.

Chapter 6 presents a model called the *Reduced Probabilistic Neural Network*, which is a radial basis function neural network used for classification that removes most of its nodes during the training phase. This chapter was published in

Wilson, D. Randall, and Tony R. Martinez, (1997). "Improved Center Point Selection for Radial Basis Function Networks," In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97)*.

Chapter 7 surveys much of the work done in instance reduction techniques in the fields of statistics, pattern recognition, machine learning, and artificial intelligence. It also presents several new pruning techniques that achieve higher average accuracy than previous models while reducing storage more than most previous models.

Three of the new pruning techniques were first published in

Wilson, D. Randall, and Tony R. Martinez, (1997). "Instance Pruning Techniques," To appear in Fisher, D., ed., *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, Morgan Kaufmann Publishers, San Francisco, CA.

Work done in the above paper was extended in Chapter 7 with several additional pruning techniques and with more detail in the survey of related work, including empirical comparisons with many existing techniques. The reference for the extended paper which appears as Chapter 7 is

Wilson, D. Randall, and Tony R. Martinez, (1997). "Reduction Techniques for Exemplar-Based Learning Algorithms," submitted to *Machine Learning Journal.*

Chapter 6

Improved Center Point Selection for Probabilistic Neural Networks

"All probabilities are 50%—Either a thing will happen, or it won't."

In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97)*, 1997.

Abstract

Probabilistic Neural Networks (PNN) typically learn more quickly than many neural network models and have had success on a variety of applications. However, in their basic form, they tend to have a large number of hidden nodes. One common solution to this problem is to keep only a randomly selected subset of the original training data in building the network. This paper presents an algorithm called the Reduced Probabilistic Neural Network (RPNN) that seeks to choose a better-than-random subset of the available instances to use as center points of nodes in the network. The algorithm tends to retain non-noisy border points while removing nodes with instances in regions of the input space that are highly homogeneous. In experiments on 22 datasets, the RPNN had better average generalization accuracy than two other PNN models, while requiring an average of less than one-third the number of nodes.

1. Introduction

Probabilistic Neural Networks (PNN) [Specht, 1992] often learn more quickly than many neural network models such as backpropagation networks [Rumelhart, 1986], and have had success on a variety of applications. PNN's are a special form of radial basis function (RBF) network [Wasserman, 1993] used for classification.

The network learns from a *training set* T, which is a collection of examples called *instances*. Each instance i has an input vector $\boldsymbol{y}_i$, and an output class, denoted as $class_i$. During execution, the network receives additional input vectors, denoted as $\boldsymbol{x}$, and outputs the class that $\boldsymbol{x}$ seems most likely to belong to.

The probabilistic neural network used in this paper is shown in Figure 1. The first (leftmost) layer contains one input node for each input attribute in an application. All connections in the network have a weight of 1, which means that the input vector is passed directly to each hidden node.

There is one hidden node for each training instance i in the training set. Each hidden node h_i has a center point $\boldsymbol{y}_i$ associated with it, which is the input vector of instance i. A hidden node also has a *spread factor*, σ_i, which determines the size of its *receptive field*. There are a variety of ways to set this parameter. In this paper, we set σ_i equal to a fraction f of the distance to the nearest neighbor of each instance i. The value of f begins at 0.5 and a binary search is performed to fine-tune this value.

At each of five steps the value of f that results in the highest average confidence of classification is chosen.

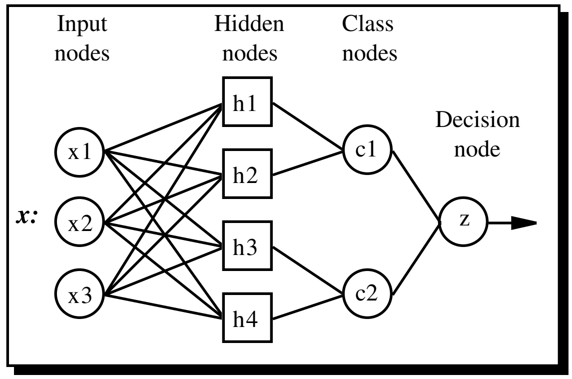


Figure 1. Probabilistic Neural Network.

A hidden node receives an input vector $\boldsymbol{x}$ and outputs an activation given by the Gaussian function g, which returns a value of 1 if $\boldsymbol{x}$ and $\boldsymbol{y}_i$ are equal, and drops to an insignificant value as the distance grows:

$$g(\boldsymbol{x}, \boldsymbol{y}_i, \sigma_i) = \exp[-D^2(\boldsymbol{x}, \boldsymbol{y}_i) / 2\sigma_i{}^2] \tag{1}$$

The distance function D determines how far apart the two vectors are. By far the most common distance function used in PNN's is Euclidean distance. However, in order to appropriately handle applications that have both linear and nominal attributes, we use a heterogeneous distance function *HVDM* [Wilson & Martinez, 1996, 1997] that uses normalized Euclidean distance for linear attributes and the Value Difference Metric (VDM) [Stanfill & Waltz, 1986] for nominal attributes. It is defined as follows.

$$HVDM(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{a=1}^{m} d_a(x_a, y_a)^2} \tag{2}$$

where m is the number of attributes. The function $d_a(x,y)$ returns a distance between the two values x and y for attribute a and is defined as

$$d_a(x, y) = \begin{cases} 1 & \text{if } x \text{ or } y \text{ is unknown} \\ vdm_a(x, y) & \text{if } a \text{ is nominal} \\ diff_a(x, y) & \text{if } a \text{ is linear} \end{cases} \tag{3}$$

The function $d_a(x,y)$ uses the following function, based on the Value Difference Metric (VDM) [Stanfill & Waltz, 1986] for nominal (discrete, unordered) attributes:

$$vdm_a(x, y) = \sqrt{\sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^2} \tag{4}$$

where $N_{a,x}$ is the number of times attribute a had value x; $N_{a,x,c}$ is the number of times attribute a had value x and the output class was c; and C is the number of output classes. For linear attributes the following function is used.

$$diff_a(x,y) = \frac{|x-y|}{4s_a} \tag{5}$$

where s_a is the sample standard deviation of the values occurring for attribute a in the training set.

Each hidden node h_i in the network is connected to a single class node. If the output class of instance i is j, then h_i is connected to class node c_j. Each *class node* c_j computes the sum of the activations of the hidden nodes that are connected to it (i.e., all the hidden nodes for a particular class) and passes this sum to a decision node. The *decision node* outputs the class with the highest summed activation.

One of the greatest advantages of this network is that it does not require any iterative training, and thus can learn quite quickly. However, one of the main disadvantages of this network is that it has one hidden node for each training instance and thus requires more computational resources (storage and time) during execution than many other models. When simulated on a serial machine, O(n) time is required to classify a single input vector. On a parallel system, only O(log n) time is required, but n nodes and nm connections are still required (where n is the number of instances in the training set, and m is the number of input attributes).

The most direct way to reduce storage requirements and speed up execution is to reduce the number of nodes in the network. One common solution to this problem is to keep only a randomly selected subset of the original training data in building the network. However, arbitrarily removing instances can reduce generalization accuracy. In addition, it is difficult to know how many nodes can be safely removed without a reasonable stopping criterion.

Other subset selection algorithms exist in linear regression theory [Rawlings, 1988], including *forward selection*, in which the network starts with no nodes and nodes are added one at a time to the network. Another method that has been used [MacQueen, 1967] is k-means clustering [Leonard, Kramer & Ungar, 1992].

This paper presents an algorithm called the Reduced Probabilistic Neural Network (RPNN) that begins with all of the available training instances as node centers and selectively removes them one at a time until classification accuracy suffers. The algorithm tends to retain only non-noisy border points while removing nodes with instances in regions of the input space that are highly homogeneous. The next section gives details of this algorithm.

2. Reduction Algorithm

The Reduced Probabilistic Neural Network (RPNN) begins with one node per training instance, as does the original PNN, and then uses the following basic rule to determine which nodes are removed from the network.

> *Remove a node if it does not cause more instances in the original training set to be classified incorrectly by the nodes remaining in the network.*

In other words, if the removal of a node does not hurt classification, remove it. When applying this rule to the network, the order of removal is important. In particular, it may be desirable to remove instances far from decision boundaries first, since they have the least effect on decisions. RPNN does this by finding the distance of every instance from its nearest *enemy*, which is the nearest neighbor of a different class, and then sorting the instances by that distance. The above rule is then applied beginning with the node furthest from its nearest enemy and proceeding to that which is closest to its nearest enemy.

In order to decide if the removal of a node degrades classification accuracy, each instance in the original training set is queried to see if its classification would be altered by the removal of the instance in question.

Specifically, in our serial implementation each instance maintains a vector of activations with one activation level for each class. The removal of a particular node would subtract some amount of activation (dependent on the distance) from its own class if removed. In addition, if the removed instance I is the nearest neighbor of some other instance A, then A must find a new nearest neighbor, and update its σ accordingly, which in turn changes what effect A has on all other instances.

The change in activation due to both the removal of I and the possible change in σ of other nodes may be enough to cause the classification of some instances to change. The change can cause an instance that used to be correctly classified to be misclassified, or cause an instance that was misclassified to be correctly classified. Such changes are counted, and if the number of newly misclassified instances is less than or equal to the number of new correctly classified instances, then the removal is performed, and the changes in activation values and σ parameters are made permanent. Otherwise they are restored to their previous values.

In order to reduce the effect of noisy instances on the network, the instance corresponding to the node that is being considered for removal is not included in the tabulation. This means a node can be removed even if its instance is itself no longer classified correctly, as long as other instances are not hurt.

To further reduce the effect of noise, a noise-reduction pass through the network is done first, beginning with the instance closest to its nearest enemy, since noisy instances are often close to instances of another class. During the noise reduction step, the criteria for removal is more strict. In order to be removed, an instance must not hurt classification, as explained above, and it must also strictly increase the average *confidence* of classification. The confidence for each node is defined as the activation of the correct class divided by the sum of activations for all of the output classes.

Noisy instances are often located near instances of another class but far from instances of their own class, so their removal will increase confidence of nearby instance's classification while having a much smaller effect on instances of their own class. Other instances, however, will typically lower confidence of nearby neighbors, which are largely of the same class, while having a smaller effect on instances of different classes. Therefore, the test of confidence is appropriate during the noise-reduction pass, but would prevent almost any pruning from taking place if used in the remainder of the algorithm.

3. Empirical Results

The Reduced Probabilistic Neural Network (RPNN) algorithm was implemented and tested on 22 applications from the Machine Learning Database Repository at the University of California, Irvine [Merz & Murphy, 1996].

Each test consisted of ten trials. Each trial consisted of learning from 90% of the training instances, and then seeing how many of the remaining 10% of the instances were classified correctly.

The RPNN was compared to EPNN, a standard Probabilistic Neural Network (PNN) that retains 100% of the instances in the training set and uses a normalized Euclidean distance metric with σ set to the distance of a node's instance to its nearest neighbor. The RPNN was also compared to HPNN, a PNN that uses the same heterogeneous distance function HVDM as RPNN, but retains 100% of the instances. HPNN and RPNN both used a dynamically adjusted spreading factor, as explained in Section 1, and RPNN used only a subset of the available instances for generalization.

Table 1 summarizes the empirical results. For each database the table shows the average accuracy for the EPNN and HPNN using all of the instances, and for the RPNN, using the percentage of instances shown.

Dataset	EPNN	HPNN	RPNN	*(size)*
Anneal	76.2	76.2	94.9	*38.3*
Audiology	36.0	57.0	54.0	*38.9*
Australian	80.1	83.8	79.7	*27.5*
Breast Cancer (WI)	97.0	94.7	92.9	*20.0*
Bridges	52.4	55.2	57.3	*21.4*
Crx	75.4	84.4	82.3	*27.2*
Echocardiogram	78.0	76.8	90.9	*9.0*
Flag	45.7	53.6	47.0	*35.7*
Heart (Hungarian)	64.0	66.7	80.3	*25.1*
Heart (More)	46.0	68.8	71.8	*21.3*
Heart	80.7	81.5	73.3	*24.8*
Heart (Swiss)	38.9	93.5	78.3	*1.4*
Hepatitis	79.3	80.6	77.3	*15.3*
Horse-Colic	67.1	67.1	68.7	*17.8*
Iris	94.0	91.3	94.7	*44.2*
Liver-Bupa	62.5	66.6	57.6	*29.2*
Pima-Indians-Diabetes	76.3	74.1	67.2	*30.3*
Promoters	54.3	84.5	88.7	*52.5*
Soybean-Large	13.0	13.0	49.9	*51.5*
Vowel	92.0	92.4	84.7	*65.2*
Wine	94.4	97.2	92.2	*40.7*
Zoo	78.9	71.1	82.2	*26.1*
Average	**67.4**	**74.1**	**75.7**	***30.2***

Table 1. Generalization Accuracy of PNN and RPNN.

The last line of Table 1 shows that RPNN had the highest average accuracy over all 22 datasets of the three algorithms, while using less than one-third of the instances (on average) for generalization. RPNN's average accuracy was slightly higher than HPNN, and both of these were substantially higher than EPNN, due in part to the use of the HVDM distance function.

Using a dynamically adjusted spreading factor had very little effect on the accuracy of HPNN (less than 1% on average), but resulted in a large improvement on RPNN (75.5% accuracy instead of 71.5%) as well as improved size reduction.

The success of RPNN varies depending upon the application. For example, on the *Vowel* dataset, it retained almost two-thirds of the instances while suffering a large drop in accuracy compared to the other two models. However, in the *Echocardiogram* dataset, the RPNN used only 9% of the data while improving generalization accuracy by over 12%. Future research will focus on identifying characteristics of applications that help determine whether the RPNN model is appropriate.

It should be noted that these datasets are not especially large (only a few hundred instances in most cases), and that the reduction in size can be even more dramatic when there are more instances available. This is especially true when the number of instances is large compared to the complexity of the decision surface.

4. Conclusion

The Reduced Probabilistic Neural Network (RPNN) reduces the size and execution time of a PNN by removing nodes from the network that are estimated to be least needed for proper generalization. It tends to retain non-noisy border points in the input space while removing nodes that are either noisy or have centers that are far from the decision boundaries. By so doing, it can fairly quickly find a reasonable subset of nodes to include in the PNN, thus reducing network complexity and execution time, as well as reducing sensitivity to noise.

The RPNN requires $O(n^2)$ time for learning on a serial machine, but only $O(n\log n)$ time in a parallel network, and in our experiments on 22 datasets reduced storage by over two-thirds on average.

It is possible that the RPNN could achieve even higher size reduction as well as more robust accuracy by employing search techniques such as genetic algorithms after initial pruning. Such search techniques could find additional nodes to remove, fine tune the spreading factor of individual nodes, and even adjust the nodes' center points. Future research will address this question, and continue to seek improved size reduction techniques. The results of this study are encouraging and show the potential for substantial reduction without sacrificing generalization ability.

References

Leonard, J. A., M. A. Kramer, and L. H. Ungar, "Using Radial Basis Functions to Approximate a Function and Its Error Bounds," *IEEE Transactions on Neural Networks*, 3, 4, pp. 624-627, 1992

MacQueen, J., "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, Berkeley, CA, pp. 281-297, 1967.

Merz, C. J., and P. M. Murphy, *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science, 1996. Internet: http://www.ics.uci.edu/~mlearn/ MLRepository.html.

Rawlings, J. O., *Applied Regression Analysis*. Wadsworth & Brooks/Cole, Pacific Grove, CA, 1988.

Rumelhart, D. E., and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, 1986.

Specht, Donald F., “Enhancements to Probabilistic Neural Networks,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN ’92)*, 1, pp. 761-768, 1992.

Stanfill, C., and D. Waltz, “Toward memory-based reasoning,” *Communications of the ACM*, 29, 1986.

Wasserman, Philip D., *Advanced Methods in Neural Computing*, New York, NY: Van Nostrand Reinhold, pp. 147-176, 1993.

Wilson, D. Randall, and Tony R. Martinez, “Heterogeneous Radial Basis Functions,” *Proceedings of the International Conference on Neural Networks (ICNN’96)*, 2, pp. 1263-1267, 1996.

Wilson, D. Randall, and Tony R. Martinez, “Improved Heterogeneous Distance Functions,” *Journal of Artificial Intelligence Research (JAIR)*, 6, 1, pp. 1-34, 1997.

Chapter 7

Reduction Techniques for Exemplar-Based Learning Algorithms

"And the bad shall be cast away,yea, even out of all the land of my vineyard; for behold, only this once will I prune my vineyard."
—Jacob 5:69

Submitted to *Machine Learning Journal*, 1997.

A shorter version of this chapter appears in
Wilson, D. Randall, and Tony R. Martinez, "Instance Pruning Techniques,"
To appear in Fisher, D., ed., *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, Morgan Kaufmann Publishers,
San Francisco, CA, July 1997.

Abstract

Exemplar-based learning algorithms are often faced with the problem of deciding which instances or other exemplars to store for use during generalization. Storing too many exemplars can result in large memory requirements and slow execution speed, and can cause an oversensitivity to noise. This paper has two main purposes. First, it provides a survey of existing algorithms used to reduce the number of exemplars retained in exemplar-based learning models. Second, it proposes six new reduction algorithms called DROP1-5 and DEL that can be used to prune instances from the concept description. These algorithms and 10 algorithms from the survey are compared on 31 datasets. Of those algorithms that provide substantial storage reduction, the DROP algorithms have the highest generalization accuracy in these experiments, especially in the presence of noise.

1. Introduction

In supervised learning, a machine learning model is shown a *training set*, *T*, which is a collection of training examples called *instances*. Each instance has an input vector and an output value. After learning from the training set, the learning model is presented with additional input vectors, and the model must *generalize*, i.e., it must use some *bias* [Mitchell, 1980; Schaffer, 1993; Dietterich, 1989; Wolpert, 1993] decide what the output value should be even if the new input vector was not in the training set.

During generalization, a large number of machine learning models compute a distance between the input vector and stored *exemplars*. Exemplars can be instances from the original training set, or can be in other forms such as hyperrectangles, prototypes, or rules. Many such *exemplar-based* learning models exist, and they are often faced with the problem of deciding how many exemplars to store and what portion of the instance space they should cover.

One of the most straightforward exemplar-based learning algorithms is the *nearest neighbor* algorithm [Cover & Hart, 1967; Hart, 1968; Dasarathy, 1991]. In the nearest neighbor and other i*nstance-based learning* (IBL) algorithms [Aha, Kibler & Albert, 1991; Aha, 1992], the exemplars are original instances from the training set. During generalization, these systems use a distance function to determine how close a new input vector ***y*** is to each stored instance, and use the nearest instance or instances to predict the output class of ***y*** (i.e., to *classify* ***y***).

Other exemplar-based machine learning paradigms include *memory-based reasoning* [Stanfill & Waltz, 1986], *exemplar-based generalization* [Salzberg, 1991; Wettschereck & Dietterich, 1995], and *case-based reasoning* (CBR) [Watson & Marir, 1994]. Such algorithms have had much success on a wide variety of domains. There are also several exemplar-based neural network learning models, including probabilistic neural networks (PNN) [Specht, 1992; Wilson & Martinez, 1996, 1997b] and other radial basis function networks [Broomhead & Lowe, 1988; Renals & Rohwer, 1989; Wasserman, 1993], as well as counterpropagation networks [Hecht-Nielsen, 1987], ART [Carpenter & Grossberg, 1987], and competitive learning [Rumelhart & McClelland, 1986].

Exemplar-based learning models must often decide what exemplars to store for use during generalization, in order to avoid excessive storage and time complexity, and possibly to improve generalization accuracy by avoiding noise and overfitting.

For example, the basic nearest neighbor algorithm retains all of the training instances. It learns very quickly because it need only read in the training set without much further processing, and it generalizes accurately for many applications. However, since the basic nearest neighbor algorithm stores all of the training instances, it has relatively large memory requirements. It must search through all available instances to classify a new input vector, so it is slow during classification. Also, since it stores every instance in the training set, noisy instances (i.e., those with errors in the input vector or output class, or those not representative of typical cases) are stored as well, which can degrade generalization accuracy.

Techniques such as *k-d trees* [Sproull, 1991] and *projection* [Papadimitriou & Bentley, 1980] can reduce the time required to find the nearest neighbor(s) of an input vector, but they do not reduce storage requirements, nor do they address the problem of noise. In addition, they often become much less effective as the dimensionality of the problem (i.e., the number of input attributes) grows.

On the other hand, when some of the instances are removed from the training set, the storage requirements and time necessary for generalization are correspondingly reduced. This paper focuses on the problem of reducing the size of the stored set of instances (or other exemplars) while trying to maintain or even improve generalization accuracy. It accomplishes this by first providing a relatively thorough survey of machine learning algorithms used to reduce the number of exemplars needed by learning algorithms, and then by proposing several new reduction techniques.

Section 2 discusses several issues related to the problem of instance set reduction, and provides a framework for discussion of individual reduction algorithms. Section 3 surveys much of the work done in this area. Section 4 presents a collection of six new algorithms called *DROP1-6* and *DEL* that are used to reduce the size of the training set while maintaining or even improving generalization accuracy. Section 5 presents empirical comparing 10 of the surveyed techniques with the six new techniques on 31 datasets. Section 6 provides conclusions and future research directions.

2. Issues in Exemplar Set Reduction

This section provides a framework for the discussion of the exemplar reduction algorithms presented in later sections. The issues discussed in this section include exemplar representation, the order of the search, the choice of distance function, the general intuition of which instances to keep, and how to evaluate the different reduction strategies.

2.1. Representation

One choice in designing a training set reduction algorithm is to decide whether to retain a subset of the original instances or whether to modify the instances using a new representation. For example, some models [Salzberg, 1991; Wettschereck & Dietterich, 1995] use hyperrectangles to represent collections of instances; instances can be generalized into rules [Domingos, 1995]; and prototypes can be used to represent a cluster of instances [Chang, 1974], even if no original instance occurred at the point where the prototype is located.

On the other hand, many models seek to retain a subset of the original instances. One problem with using the original data points is that there may not be any data points located at the precise points that would make for the most accurate and concise concept description. Prototypes, on the other hand, can be artificially constructed to exist exactly where they are needed, if such locations can be accurately determined.

Similarly, rules and hyperrectangles can be constructed to reduce the need for instances in certain areas of the input space. However they are often restricted because of their axis-aligned boundaries.

2.2. Direction of Search

When searching for a subset *S* of instances to keep from training set *T*, there are also a variety of directions the search can proceed, including *incremental, decremental,* and *batch*.

2.2.1. Incremental

The incremental search begins with an empty subset *S*, and adds each instance in *T* to *S* if it fulfills some criteria. In this case the order of presentation of instances can be very important. In particular, the first few instances may have a very different probability of being included in *S* than they would if they were visited later.

Under such schemes, the order of presentation of instances in *T* to the algorithm is typically random because by definition, an incremental algorithm should be able to handle new instances as they are made available without all of them being present at the beginning. In addition, some incremental algorithms do not retain all of the previously seen instances even during the learning phase, which can also make the order of presentation important.

One advantage to an incremental scheme is that if instances are made available later, after training is complete, they can continue to be added to *S* according to the same criteria. The main disadvantage is that incremental algorithms are sensitive to the order of presentation of the instances, and their early decisions are based on very little information, and are therefore prone to errors until more information is available.

It should be noted that some algorithms add instances to *S* in a somewhat incremental fashion, but they examine all available instances to help select which instance to add next. This makes the algorithm not truly incremental, but may improve its performance substantially.

2.2.2. Decremental

The decremental search begins with $S=T$, and then searches for instances to remove from S. Again the order of presentation is important, but unlike the incremental process, all of the training examples are available for examination at any time, so a search can be made to determine which instance would be best to remove during each step of the algorithm. Decremental algorithms discussed in Section 3 include RNN [Gates, 1972], SNN [Ritter et al., 1975], ENN [Wilson, 1972], VSM [Lowe, 1995], and the Shrink (Subtractive) Algorithm [Kibler & Aha, 1987]. NGE [Salzberg, 1991] and RISE [Domingos, 1995] can also be viewed as decremental algorithms, except that instead of simply removing instances from S, they are generalized into hyperrectangles or rules. Similarly, Chang's prototype rule [Chang, 1974] operates in a decremental order, but prototypes are merged into each other instead of being simply removed.

One disadvantage with the decremental rule is that it is often computationally more expensive than incremental algorithms. For example, in order to find the nearest neighbor in T of an instance, n distance calculations must be made. On the other hand, there are fewer than n instances in S (zero initially, and some fraction of T eventually), so finding the nearest neighbor in S of an instance takes less computation.

However, if the application of a decremental algorithm can result in greater storage reduction, then the extra computation during learning (which is done just once) can be well worth the computational savings during execution thereafter. Increased generalization accuracy, if it can be achieved, is also typically worth some extra time during learning.

2.2.3. Batch

Another way to apply a training set reduction rule is in batch mode. This involves deciding if each instance meets the removal criteria before removing any of them. Then all those that do meet the criteria are removed at once. For example, the *All-kNN* rule [Tomek, 1976] operates this way. This can relieve the algorithm from having to constantly update lists of nearest neighbors and other information when instances are individually removed.

However, there are also dangers in batch processing. For example, assume the following rule is applied to an instance set.

> *Remove an instance if it has the same output class as its k nearest neighbors.*

This could result in entire clusters disappearing if there are no instances of a different class nearby. If done in decremental mode, however, some instances would remain, because eventually enough neighbors would be removed that one of the k nearest neighbors of an instance would have to be of another class, even if it was originally surrounded by those of its own class.

As with decremental algorithms, batch processing suffers from increased time complexity over incremental algorithms.

2.3. Border points vs. central points

Another factor that distinguishes instance reduction techniques is whether they seek to retain border points, central points, or some other set of points.

The intuition behind retaining border points is that "internal" points do not affect the decision boundaries as much as border points, and thus can be removed with relatively little effect on classification.

On the other hand, some algorithms instead seek to *remove* border points. They remove points that are noisy or do not agree with their neighbors. This removes close border points, leaving smoother decision boundaries behind. However, such algorithms do not remove internal points that do not necessarily contribute to the decision boundary.

It may take a large number of border points to completely define a border, so some algorithms retain *center* points in order to use those instances which are most typical of a particular class to classify instances near them. This can dramatically affect decision boundaries, because the decision boundaries depend on not only where the instances of one class lie, but where those of other classes lie as well. In general, the decision boundary lies halfway between two nearest instances of opposing classes, so center points must be chosen carefully in order to keep the decision boundaries in the correct general vicinity.

2.4. Distance Function

The distance function (or its complement, the similarity function) used to decide which neighbors are closest to an input vector can have a dramatic effect on an exemplar-based learning system.

The nearest neighbor algorithm and its derivatives usually use the Euclidean distance function, which is defined as

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

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are the two input vectors, m is the number of input attributes, and x_i and y_i are the input values for input attribute i. This function is appropriate when all the input attributes are numeric and have ranges of approximately equal width. When the attributes have substantially different ranges, the attributes can be normalized by dividing the individual attribute distances by the range or standard deviation of the attribute.

A variety of other distance functions are also available for continuously valued attributes, including the Minkowsky [Batchelor, 1978], Mahalanobis [Nadler & Smith, 1993], Camberra, Chebychev, Quadratic, Correlation, and Chi-square distance metrics [Michalski, Stepp & Diday, 1981; Diday, 1974]; the Context-Similarity measure [Biberman, 1994]; the Contrast Model [Tversky, 1977]; hyperrectangle distance functions [Salzberg, 1991; Domingos, 1995] and others. Several of these functions are defined in Figure 1.

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r\right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T Q(\boldsymbol{x}-\boldsymbol{y}) = \sum_{j=1}^{m}\left(\sum_{i=1}^{m}(x_i - y_i)q_{ji}\right)(x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m}(\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}} V^{-1}(\boldsymbol{x}-\boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m}(x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m}(x_i - \overline{x_i})^2 \sum_{i=1}^{m}(y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i}\left(\frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}}\right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)}\sum_{i=1}^{m}\sum_{j=1}^{i-1}\operatorname{sign}(x_i - x_j)\operatorname{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure 1. Equations of selected distance functions ($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

When *nominal* (discrete, unordered) attributes are included in an application, a distance metric is needed that supports them. Some learning models have used the *overlap* metric, which defines the distance for an attribute as 0 if the values are equal, or 1 if they are different, regardless of which two values they are.

An alternative distance function for nominal attributes is the Value Difference Metric (VDM) [Stanfill & Waltz, 1986]. Using the VDM, the distance between two values x and y of a single attribute a is given as

$$vdm_a(x,y) = \sum_{c=1}^{C}\left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}}\right)^2 \qquad (2)$$

where $N_{a,x}$ is the number of times attribute a had value x; $N_{a,x,c}$ is the number of times attribute a had value x and the output class was c; and C is the number of output classes. Using this distance measure, two values are considered to be closer if they have more similar classifications, regardless of the order of the values.

In order to handle heterogeneous applications—those with both numeric and nominal attributes—it is possible to use a heterogeneous distance function such as *HVDM* [Wilson & Martinez, 1997], which is defined as

$$HVDM(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{a=1}^{m} d_a(x_a, y_a)^2} \tag{3}$$

where the function $d_a(x,y)$ is the distance for attribute a and is defined as

$$d_a(x,y) = \begin{cases} 1 & \text{if } x \text{ or } y \text{ is unknown; otherwise...} \\ vdm_a(x,y) & \text{if } a \text{ is nominal} \\ \dfrac{|x-y|}{4\sigma_a} & \text{if } a \text{ is numeric} \end{cases} \tag{4}$$

where $vdma(x,y)$ is the function given in (2), and σ_a is the standard deviation of the values occurring for attribute a in the instances in the training set T. This function handles unknown input values by assigning them a large distance, and provides appropriate normalization between numeric and nominal attributes, as well as between numeric attributes of different scales.

Several algorithms use weighting schemes that alter the distance measurements and voting influence of each instance. In this paper we focus on training set reduction, and thus will not use any weighting schemes in our experiments other than those needed for normalization in the distance function, as explained above. A good survey of weighting schemes is given by Wettschereck, Aha and Mohri [1995].

2.5. Voting

Another decision that must be made for many algorithms is the choice of k, which is the number of neighbors used to decide the output class of an input vector. The value of k is typically a small, odd integer (e.g., 1, 3 or 5). Usually each such nearest neighbor gets exactly one vote, so even values of k could result in “ties” that would have to be resolved arbitrarily or through some more complicated scheme. There are some algorithms which give closer neighbors more influence than further ones, such as the Distance-Weighted *k*NN Rule [Dudani, 1976]. Such modifications reduce the sensitivity of the algorithm to the selection of k. Radial Basis Function networks [Wasserman, 1993] and Probabilistic Neural Networks [Specht, 1992] use a Gaussian weighting of influence and allow all instances to “vote”, though instances that are very far from the input have only negligible influence. This does away with the need for the k parameter, but introduces a need for weight-spreading parameters.

One common way of determining the value of k is to use *leave-one-out cross-validation*. For each of several values of k, each instance is classified by its k nearest neighbors other than the instance itself, to see if it is classified correctly. The value for k that produces the highest accuracy is chosen.

In the basic nearest neighbor rule, setting k greater than 1 decreases the sensitivity of the algorithm to noise, and tends to smooth the decision boundaries somewhat. It is also important for many pruning algorithms to have a $k > 1$. However, once pruning has taken place, it is possible that the value of k should be changed. For example, if the training set has been reduced to the point that there is only one instance representing what was formerly a cluster of instances, then perhaps $k = 1$ would be more appropriate than $k > 1$, especially if the noisy instances have been removed during the reduction process. In other cases, the value of k should remain the same. Thus, it may be appropriate to find a value of k for use during the pruning process, and then redetermine the best value for k after pruning is completed.

It may even be advantageous to update k dynamically during the pruning process. For example, if a very large value of k were used initially, the order of removal of instances from the subset might be improved.

2.6. Evaluation Strategies

In comparing training set reduction algorithms, there are a number of criteria that can be used to compare the relative strengths and weaknesses of each algorithm. These include speed increase (during execution), storage reduction, noise tolerance, generalization accuracy, time requirements (during learning), and incrementality.

2.6.1. Storage reduction

One of the main goals of training set reduction algorithms is to reduce storage requirements. It is important to note that if alternate representations are used (e.g., hyperrectangles or rules), any increase in the size of the new representation must be taken into account along with the reduction in number of instances stored.

2.6.2. Speed increase

Another main goal is to speed up classification. A reduction in the number of stored instances will typically yield a corresponding reduction in the time it takes to search through these instances and classify a new input vector. Again, more complex representations such as hyperrectangles may not need as many comparisons, but may require more computation for each comparison, and this must be taken into account.

2.6.3. Generalization accuracy

A successful algorithm will be able to significantly reduce the size of the training set without significantly reducing generalization accuracy. Ideally, algorithms will be able to detect at what point generalization accuracy will suffer, and reduce the size of the subset to near that point. In some cases generalization accuracy can increase with the reduction of instances, such as when noisy instances are removed and when decision boundaries are smoothed to more closely match the true underlying function rather than the sampling distribution.

2.6.4. Noise tolerance

Algorithms also differ with respect to how well they work in the presence of noise. In the presence of noise, there are two main problems that can occur. The first is that very few instances will be removed from the training set because they are needed to maintain the noisy (and thus overly complex) decision boundaries. The second problem is that generalization accuracy can suffer, especially if noisy instances are retained while good instances are removed. In such cases the reduced training set can be much less accurate than the full training set in classifying new input vectors.

2.6.5. Learning speed

The learning process is done just once on a training set, so it is not quite as important for the learning phase to be fast. However, if the learning phase takes too long it can become impractical for real applications. Ironically, it is on especially large training sets that reduction algorithms are most badly needed, so a reasonable (e.g., $O(n^2)$ or faster) time bound is desirable.

2.6.6. Incremental

In some cases it is convenient to have an incremental algorithm so that additional instances can be added over time as they become available. On the other hand, it is possible to use a non-incremental algorithm on an initial database and then employ a separate incremental database once a reasonable starting point has been achieved.

Note that not all algorithms attempt to meet all of these goals. For example, a hybrid hyperrectangle and nearest-neighbor algorithm by Wettschereck and Dietterich [1995] saves all of the training set in addition to the hyperrectangles, and thus actually increases storage requirements. However, it uses the hyperrectangles to quickly classify most input vectors, and only uses the entire training set when necessary. Thus, it sacrifices the goal of storage reduction in favor of the goals of classification speed and maintaining or increasing generalization accuracy.

3. Survey of Instance Reduction Algorithms

Many researchers have addressed the problem of training set size reduction. This section surveys several techniques, discusses them in light of the framework presented in Section 2, and points out their interesting differences. Most of the models discussed here use a subset of the original instances as their representation, and though most have primarily used the Euclidean distance function in the past, they can typically make use of the *HVDM* distance function or other distance functions when needed. Most of the models also tend to use $k = 1$ except where noted, though in most cases the algorithms can be modified to use $k > 1$.

In order to avoid repeating lengthy definitions, some notation is introduced here. A training set T consists of n instances (or prototypes) $P_{1..n}$. Each instance P has k nearest neighbors $P.N_{1..k}$ (ordered from nearest to furthest), where k is typically a small odd integer such as 1, 3 or 5. P also has a nearest *enemy*, $P.E$, which is the nearest instance with a different output class. Those instances that have P as one of their k nearest neighbors are called *associates* of P, and are notated as $P.A_{1..a}$ (sorted from nearest to furthest) where a is the number of associates that P has.

3.1. Nearest Neighbor Editing Rules

3.1.1. Condensed Nearest Neighbor Rule

Hart [1968] made one of the first attempts to reduce the size of the training set with his *Condensed Nearest Neighbor Rule* (CNN). His algorithm finds a subset S of the training set T such that every member of T is closer to a member of S of the same class than to a member of S of a different class. In this way, the subset S can be used to classify all the instances in T correctly.

This algorithm begins by randomly selecting one instance belonging to each output class from T and putting them in S. Then each instance in T is classified using only the instances in S. If an instance is misclassified, it is added to S, thus ensuring that it will be classified correctly. This process is repeated until there are no instances in T that are misclassified. This algorithm ensures that all instances in T are classified correctly, though it does not guarantee a minimal set.

This algorithm is especially sensitive to noise, because noisy instances will usually be misclassified by their neighbors, and thus will be retained. This causes two problems. First, storage reduction is hindered, because noisy instances are retained, and because they are there, often non-noisy instances nearby will also need to be retained. The second problem is that generalization accuracy is hurt because noisy instances are usually exceptions and thus do not represent the underlying function well. Since some neighbors have probably been pruned, a noisy instance in S will often cover more of the input space than it did in T, thus causing even more misclassifications than before reduction.

3.1.2. Selective Nearest Neighbor Rule

Ritter et. al. [1975] extended the condensed NN method in their *Selective Nearest Neighbor Rule* (SNN) such that every member of T must be closer to a member of S of the same class than to any member of T (instead of S) of a different class. Further, the method ensures a minimal subset satisfying these conditions.

The algorithm for SNN is more complex than most other reduction algorithms, and the learning time is significantly greater, due to the manipulation of an $n \times n$ matrix and occasional recursion. The SNN algorithm begins by constructing a binary $n \times n$ matrix A (where n is the number of instances in T), where A_{ij} is set to 1 when instance j is of the same class as instance i, and it is closer to instance i than i's nearest *enemy*, i.e., the nearest neighbor of i in T that is of a different class than i. A_{ii} is always set to 1.

Once this array is set up, the following 5 steps are taken until no columns remain in the array:

1. For all columns i that have exactly one bit on, let j be the row with the bit on in column i. Row j is removed, instance j is added to S, and all columns with a bit on in row j are removed.

2. For all rows j, delete row j if for all (remaining) columns i and for some (remaining) row k, $A_{ji} \le A_{ki}$. In other words, row j is deleted if, whenever row j contains a 1, row k also contains a 1. In this case instance j is *not* added to S.

3. Delete any column i if for all (remaining) rows j and some (remaining) column k, $A_{ji} \geq A_{jk}$.

4. Continue to repeat steps 1-3 until no further progress can be made. If no columns remain in the array, then S is complete and the algorithm is finished. Otherwise, go on to step 5.

5. Find the row j that when included in S requires the fewest other rows to also be included in S. This is done as follows:
   (a) For each remaining row j, assume that instance j will be added to S, and that row j and any (remaining) columns with a bit on in row j will be deleted. Subject to this assumption, find the fewest number of additional rows it would take to get at least as many 1's as there are remaining columns. (Do not actually remove row j or the columns yet). From the minimums found for each row j, keep track of the absolute minimum found by any row j.
   (b) For each row j in (a) that resulted in the absolute minimum number of additional rows that *might* be needed, actually remove j and columns with bits on in row j and call the algorithm recursively beginning with step 1. If the minimum number of rows was really used, then stop, because S is complete. Otherwise, restore row j and the removed columns, and try the next possible row j.
   (c) If no row j is successful in achieving the minimum number, increment the absolute minimum and try (b) again until successful.

Note that the only steps in which instances are chosen for inclusion in S are steps 1 and 5.

This algorithm takes approximately $O(mn^2 + n^3)$ time, compared to the $O(mn^2)$ or less time required by most other algorithms surveyed. It also requires $O(n^2)$ storage during the learning phase for the matrix, though this matrix is discarded after learning is complete. The algorithm is sensitive to noise, though it will tend to sacrifice storage more than accuracy when noise is present.

3.1.3. Reduced Nearest Neighbor Rule

Gates [1972] introduced the *Reduced Nearest Neighbor Rule* (RNN). The RNN algorithm starts with S=T and removes each instance from S if such a removal does not cause any *other* instances in T to be misclassified by the instances remaining in S. It is computationally more expensive than Hart's condensed NN rule, but will always produce a subset of CNN, and is thus less expensive in terms of computation and storage during the classification stage.

Since the instance being removed is not guaranteed to be classified correctly, this algorithm is able to remove noisy instances and internal instances while retaining border points.

3.1.4. Edited Nearest Neighbor Rule

Wilson [1972] developed the *Edited Nearest Neighbor* (ENN) algorithm in which S starts out the same as T, and then each instance in S is removed if it does not agree with the majority of its k nearest neighbors (with k=3, typically). This edits out noisy instances as well as close border cases, leaving smoother decision boundaries. It also retains all internal points, which keeps it from reducing the storage requirements as much as most other reduction algorithms. The *Repeated ENN* (RENN) applies the ENN algorithm repeatedly until all instances remaining have a majority of their neighbors with the same class, which continues to widen the gap between classes and smoothes the decision boundary.

3.1.5. ALL-KNN

Tomek [1976] extended the ENN with his *All k-NN* method of editing. This algorithm works as follows. For i=1 to k, flag as bad any instance not classified correctly by its i nearest neighbors. After completing the loop all k times, remove any instances from S flagged as bad. In his experiments, RENN produced higher accuracy than ENN, and the *All k-NN* method resulted in even higher accuracy yet. As with ENN, this method can leave internal points intact, thus limiting the amount of reduction that it can accomplish. These algorithms serve more as noise filters than serious reduction algorithms.

3.1.6. VARIABLE SIMILARITY METRIC

Lowe [1995] presented a *Variable Similarity Metric* (VSM) learning system that produces a confidence level of its classifications. In order to reduce storage and remove noisy instances, an instance is removed if all k of its neighbors are of the same class, even if they are of a different class than t (in which case t is likely to be noisy). This removes noisy instances as well as internal instances, while retaining border instances. The instance is only removed, however, if its neighbors are at least 60% sure of their classification. The VSM system typically uses a fairly large k (e.g., k=10), and the reduction in storage is thus quite conservative, but it can provide an increase in generalization accuracy. Also, the VSM system used distance-weighted voting, which makes a larger value of k more appropriate.

3.2. "Instance-Based" Learning Algorithms

Aha et. al. [1991; Aha, 1992] presented a series of *instance-based* learning algorithms. IB1 (Instance Based learning algorithm 1) was simply the 1-NN algorithm, and was used as a baseline.

3.2.1. IB2

IB2 is incremental: it starts with S initially empty, and each instance in T is added to S if it is not classified correctly by the instances already in S (with the first instances always added). An early case study [Kibler & Aha, 1987] calls this algorithm the *Growth (Additive) Algorithm*. This algorithm is quite similar to Hart's condensed NN rule, except that IB2 does not seed S with one instance of each class, and does not repeat the process after the first pass through the training set. This means that IB2 will not necessarily classify all instances in T correctly.

This algorithm retains border points in S while eliminating internal points that are surrounded by members of the same class. Like the CNN algorithm, IB2 is extremely sensitive to noise, because erroneous instances will usually be misclassified, and thus noisy instances will almost always be saved, while more reliable instances are removed.

3.2.2. SHRINK (SUBTRACTIVE) ALGORITHM

Kibler & Aha [1987] also presented an algorithm that starts with $S=T$, and then removes any instances that would still be classified correctly by the remaining subset. This is similar to the Reduced Nearest Neighbor (RNN) rule, except that it only considers whether the removed instance would be classified correctly, whereas RNN considers whether the classification of other instances would be affected by the instance's removal. Like RNN and many of the other algorithms, it retains border points, but unlike RNN, this algorithm is sensitive to noise.

3.2.3. IB3

IB3 [Aha et al. 1991, Aha 1992] is another incremental algorithm that addresses IB2's problem of keeping noisy instances by retaining only *acceptable* misclassified instances. The algorithm proceeds as follows.

```
For each instance t in T
    Let a be the nearest acceptable instance in S to t.
    (if there are no acceptable instances in S, let a be a random instance in S)
    If class(a)≠class(t) then add t to S.
    For each instance s in S
        If s is at least as close to t as a is
            Then update the classification record of s
            and remove s from S if its classification record is significantly poor.
Remove all non-acceptable instance from S.
```

An instance is *acceptable* if the lower bound on its accuracy is statistically significantly higher (at a 90% confidence level) than the upper bound on its class' frequency. Similarly, an instance is dropped from S if the upper bound on its accuracy is statistically significantly lower (at a 70% confidence level) than the lower bound on its class' frequency. Other instances are kept in S during training, and then dropped at the end if they do not prove to be acceptable.

The formula for the statistical significance test is

$$\frac{p + z^2/2n \pm \sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + z^2/n} \tag{5}$$

where for the accuracy of an instance in S, n is the number of classification attempts since introduction of the instance to S (i.e., the number of times it was at least as close to t as a was), p is the accuracy of such attempts (i.e., the number of times the class matched t's class, divided by n), and z is the confidence (.9 for acceptance, .7 for dropping). For a class' frequency, p is the frequency (i.e. proportion of instances so far that are of this class), n is the number of previously processed instances, and z is the confidence (.9 for acceptance, .7 for dropping).

IB3 was able to achieve greater reduction in the number of instances stored and also achieved higher accuracy than IB2, due to its reduced sensitivity to noise on the applications on which it was tested.

3.2.4. IB4 AND IB5

In order to handle irrelevant attributes, IB4 [Aha, 1992] extends IB3 by building a set of attribute weights for each class. It requires fewer instances to generalize well when irrelevant attributes are present in a dataset. IB5 [Aha, 1992] extends IB4 to handle the addition of new attributes to the problem after training has already begun. These extensions of IB3 address issues that are beyond the scope of this paper, and are thus only mentioned here.

3.2.5. MODEL CLASS SELECTION

Brodley [1993] introduced a *Model Class Selection* (MCS) system that uses an instance-based learning algorithm (which claims to be "based loosely on IB3") as part of a larger hybrid learning algorithm. Her algorithm for reducing the size of the training set is to keep track of how many times each instance was one of the k nearest neighbors of another instance (as instances were being added to the concept description), and whether its class matched that of the instance being classified. If the number of times it was wrong is greater than the number of times it was correct then it is thrown out. This tends to avoid noise, though it uses a simpler approach than IB3.

3.2.6. TYPICAL INSTANCE-BASED LEARNING

Zhang [1992] used a different approach called the *Typical Instance Based Learning* (TIBL) algorithm, which attempted to save instances near the center of clusters rather than on the border. This can result in much more drastic reduction in storage and smoother decision boundaries, and is robust in the presence of noise.

The *typicality* of an instance is defined as the ratio of its average similarity to instances of the same class to its average similarity to instances of other classes. The similarity(x,y) of two instances x and y is defined as 1-distance(x,y), where

$$\text{distance}(x, y) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\frac{x_i - y_i}{\max_i - \min_i}\right)^2} \tag{6}$$

and m is the number of input attributes, $\max_i$ and $\min_i$ are the maximum and minimum values occurring for attribute i, respectively. For nominal attributes, the distance for that attribute is 0 if they are equal or 1 if they are different (i.e., the *overlap* metric). Each instance x has a weight w_x that is used during training and subsequent classification, but which is not used in computing typicality.

The learning algorithm proceeds as follows. Pick the most typical instance x in T-S that is incorrectly classified by the instances in S. Find the most typical instance y in T-S which causes x to be correctly classified, and add it to S. Note that x itself is *not* added at this point. Set y's weight to be 1/*typicality*(y). Repeat this process until all instances in T are classified correctly.

This strategy shows great reduction in storage, especially when the application has "graded structures" in which some instances are more typical of a class than others in a fairly continuous way. The TIBL algorithm also avoids saving noisy instances. It is pseudo-incremental, i.e., it proceeds in an incremental fashion, but it uses the entire training set to determine the typicality of each instance and the range of each input attribute.

The TIBL algorithm may have difficulty on problems with complex decision surfaces, and requires modifications to handle disjoint geometric regions that belong to the same class.

3.2.7. Random Mutation Hill Climbing

Skalak [1994] used *random mutation hill climbing* [Papadimitriou & Steiglitz, 1982] to select instances to use in *S*. The method begins with *m* randomly selected instances in *S* (where *m* is a parameter that is unfortunately not automatically selected). Then for each iteration (called a *mutation*), one randomly selected instance in *S* is removed and replaced with another randomly selected instance in *T*-*S*. If this strictly improves classification of the instances in *T*, the change is retained, otherwise it is undone. This process is repeated for *n* iterations, where *n* is another parameter provided by the user. Skalak used $n = 100$.

Since it does not determine the number *m* of instances to retain in the subset, this method only solves part of the problem.

3.2.8. Encoding Length

Cameron-Jones [1995] used an *encoding length heuristic* to determine how good the subset *S* is in describing *T*. The basic algorithm begins with a growing phase that takes each instance *i* in *T* and adds it to *S* if that results in a lower cost than not adding it. As with IB3, the growing phase can be affected by the order of presentation of the instances.

The *cost* (i.e., the value to be minimized) of the instance-based model is

$$COST(m,n,x) = F(m,n) + m\log_2(C) + F(x,n-m) + x\log_2(C-1) \tag{7}$$

where *n* is the number of instances in *T*, *m* is the number of instances in *S*, and *x* is the number of *exceptions* (i.e., the number of instances *seen so far* that are misclassified by the instances in *S*). $F(m,n)$ is the cost of encoding which *m* instances of the *n* available are retained, and is defined as

$$F(m,n) = \log^*\left(\sum_{j=0}^{m} C_j^n\right) = \log^*\left(\sum_{j=0}^{m} \frac{n!}{j!(n-j!)}\right) \tag{8}$$

where $\log^*(x)$ is the sum of the positive terms of $\log_2(x)$, $\log_2(\log_2(x))$, etc.

After all instances are seen, pruning is done, where each instance *i* in *S* is removed if doing so lowers the cost of the classifier. Cameron-Jones calls this method the "Pre/All" method, since it is not truly incremental, but to better distinguish it from other techniques in this paper, we call it the *Encoding Length Grow* (*ELGrow*) method.

The *Explore* method [Cameron-Jones, 1995] begins by growing and pruning *S* using the *ELGrow* method, and then performs 1000 mutations to try to improve the classifier. Each mutation tries adding an instance to *S*, removing one from *S*, or swapping one in *S* with one in *T*-*S*, and keeps the change if it does not increase the cost of the classifier. The generalization accuracy of the *Explore* method is quite good empirically, and its storage reduction is much better than most other algorithms.

3.3. Prototypes and Other Modifications of the Instances

Some algorithms seek to reduce storage requirements and speed up classification by modifying the instances themselves, instead of just deciding which ones to keep.

3.3.1. Prototypes

Chang [1974] introduced an algorithm in which each instance in T is initially treated as a prototype. The nearest two instances that have the same class are merged into a single prototype (using a weighted averaging scheme) that is located somewhere between the two prototypes. This process is repeated until classification accuracy starts to suffer.

This method achieved good results, though it requires modification to handle applications that have one or more nominal input attributes.

3.3.2. RISE

Domingos [1995] introduced the RISE 2.0 system which treats each instance in T as a rule in R. For each rule r in R, the nearest example n in T of the same class as r is found that is not yet covered by r. The rule r is then minimally generalized to cover n, unless that harms accuracy. This process is repeated until no rules are generalized during an entire pass through all the rules in R.

During generalization, the nearest rule to an input vector is used to provide the output class. If two rules are equally close, the one with higher generalization accuracy on the training set is used.

3.3.3. EACH

Salzberg [1991] introduced the nested generalized exemplar (NGE) theory, in which hyperrectangles are used to take the place of one or more instances, thus reducing storage requirements. The program used to implement NGE is called the *Exemplar-Aided Constructor of Hyperrectangles* (EACH). EACH seeds the system with several randomly selected instances from the training set, after which it operates incrementally. As each instance is presented, EACH finds the distance to the nearest exemplar (i.e., a point or hyperrectangle), which is 0 if the instance is inside the hyperrectangle. A point inside multiple hyperrectangles is considered to be closest to the smallest one.

When the new instance has the same class as its nearest exemplar, the exemplar is generalized (i.e., the hyperrectangle is grown) so that it also covers the new instance. When the classes are different, EACH attempts to change the shape of the second-closest exemplar so that it becomes the closest one. If it cannot do so, then the new instance becomes a new exemplar. Weights are maintained for each exemplar that reduce the effect of noisy exemplars and irrelevant attributes.

Wettschereck & Dietterich [1995] introduced a hybrid nearest-neighbor and nearest-hyperrectangle algorithm that uses hyperrectangles to classify input vectors if they fall inside the hyperrectangle, and kNN to classify inputs that were not covered by any hyperrectangle. This algorithm must store the entire training set T, but accelerates classification by using relatively few hyperrectangles whenever possible.

4. New Reduction Algorithms: DROP1-5 and DEL

Given the issues in Section 2 to consider, our research has been directed towards finding instance reduction techniques that provide noise tolerance, high generalization accuracy, insensitivity to the order of presentation of instances, and significant storage reduction, which in turn improves generalization speed.

This section presents a collection of new heuristics used to decide which instances to keep and which instances to remove from a training set. Unlike most previous methods, these algorithms take careful note of the order in which instances are removed. The first three methods, DROP1-3, were previously introduced by the authors under the name RT1-3, respectively [Wilson & Martinez, 1997c].

4.1. DROP1

The first new reduction technique we present is the *Decremental Reduction Optimization Procedure 1*, or *DROP1*. This procedure uses the following basic rule to decide if it is safe to remove an instance from the instance set S (where $S = T$ originally).

Remove P if at least as many of its associates in S
would be classified correctly without P.

To see if an instance P can be removed using this rule, each associate (i.e., each instance that has P as one of its neighbors) is checked to see what effect the removal of P would have on it.

Removing P causes each associate $P.A_i$ to use its k+1st nearest neighbor ($P.A_i.N_{k+1}$) in place of P. If P has the same class as $P.A_i$, and $P.A_i.N_{k+1}$ has a different class than $P.A_i$, this weakens its classification, and could cause $P.A_i$ to be misclassified by its neighbors. On the other hand, if P is a different class than $P.A_i$ and $P.A_i.N_{k+1}$ is the same class as $P.A_i$, the removal of P could cause a previously misclassified instance to be classified correctly.

In essence, this rule tests to see if removing P would degrade leave-one-out cross-validation generalization accuracy, which is an estimate of the true generalization ability of the resulting classifier. An instance is removed when it results in the same level of generalization with lower storage requirements. By maintaining lists of k+1 neighbors and an average of k+1 associates (and their distances), the leave-one-out cross-validation can be computed in O(k) time for each instance instead of the usual O(mn) time, where n is the number of instances in the training set, and m is the number of input attributes. An O(mn) step is only required once an instance is selected for removal.

The algorithm for DROP1 proceeds as shown in Figure 2. This algorithm begins by building a list of nearest neighbors for each instance, as well as a list of associates. Then each instance in S is removed if its removal does not hurt the classification of the instances remaining in S. When an instance P is removed, all of its associates must remove P from their list of nearest neighbors, and then must find a new nearest neighbor so that they still have k+1 neighbors in their list. When they find a new neighbor N, they also add themselves to N's list of associates so that at all times every instance has a current list of neighbors and associates.

```
DROP1(Training set T): Instance set S.
    Let S = T.
    For each instance P in S:
        Find P.N1..k+1, the k+1 nearest neighbors of P in S.
        Add P to each of its neighbors' lists of associates.
    For each instance P in S:
        Let with = # of associates of P classified correctly with P as a neighbor.
        Let without = # of associates of P classified correctly without P.
        If (without - with) ≥ 0
            Remove P from S.
            For each associate A of P
                Remove P from A's list of nearest neighbors
                Find a new nearest neighbor for A.
                Add A to its new neighbor's list of associates.
            For each neighbor N of P
                Remove P from its N's lists of associates.
        Endif
    Return S.
```

Figure 2: Pseudo-code for DROP1.

This algorithm removes noisy instances, because a noisy instance *P* usually has associates that are mostly of a different class, and such associates will be at least as likely to be classified correctly without *P*. DROP1 also removes instances in the center of clusters, because associates there are not near their enemies, and thus continue to be classified correctly without *P*.

Near the border, the removal of some instances can cause others to be classified incorrectly because the majority of their neighbors can become enemies. Thus this algorithm tends to keep non-noisy border points. At the limit, there is typically a collection of border instances such that the majority of the k nearest neighbors of each of these instances is the correct class.

4.2. DROP2: Using More Information and Ordering the Removal.

There is a potential problem that can arise in DROP1 with regards to noisy instances. A noisy instance will typically have associates of a different class, and will thus be contained to a somewhat small portion of the input space. However, if its associates are removed by the above rule, the noisy instance may cover more and more of the input space. Eventually it is hoped that the noisy instance itself will be removed. However, if many of its neighbors are removed first, its associates may eventually include instances of the same class from the other side of the original decision boundary, and it is possible that removing the noisy instance at that point could cause some of its distant associates to be classified incorrectly.

DROP2 solves this problem by considering the effect of the removal of an instance on all the instances in the *original* training set *T* instead of considering only those instances remaining in *S*. In other words, an instance *P* is removed from *S* only if at least as many of its associates—including those that may have already been pruned—are classified correctly without it.

Thus, the removal criterion can be restated as

Remove P if at least as many of its associates in T
would be classified correctly without P.

Using this modification, each instance P in the original training set T continues to maintain a list of its $k + 1$ nearest neighbors in S, even after P is removed from S. This in turn means that instances in S have associates that are both in and out of S, while instances that have been removed from S have no associates (because they are no longer a neighbor of any instance). This modification makes use of additional information that is available for estimating generalization accuracy, and also avoids some problems that can occur with DROP1 such as removing entire clusters. This change is made by removing lines 15 and 16 from the pseudo-code for DROP1 in Figure 2 so that pruned instances will still be associates of their nearest neighbors in S.

DROP2 also changes the order of removal of instances. It initially sorts the instances in S by the distance to their nearest enemy. Instances are then checked for removal beginning at the instance furthest from its nearest enemy. This tends to remove instances furthest from the decision boundary first, which in turn increases the chance of retaining border points.

4.3. DROP3: Filtering Noise.

DROP2 sorts S in an attempt to remove center points before border points. One problem with this method is that noisy instances are also "border" points, and cause the order of removal to be drastically changed. One noisy point in the center of a cluster causes many points in that cluster to be considered border points, and some of these can remain in S even after the noisy point is removed.

Two passes through S can remove the dangling center points, but unfortunately, by that time some border points may have already been removed that should have been kept.

DROP3 therefore uses a noise-filtering pass *before* sorting the instances in S. This is done using a rule similar to ENN [Wilson, 1972]: Any instance misclassified by its k nearest neighbors is removed. This removes noisy instances, as well as close border points, which can in turn smooth the decision boundary slightly. This helps to avoid "overfitting" the data, i.e., using a decision surface that goes beyond modeling the underlying function and starts to model the data sampling distribution as well.

After removing noisy instances from S in this manner, the instances are sorted by distance to their nearest enemy remaining in S, and thus points far from the real decision boundary are removed first. This allows points internal to clusters to be removed early in the process, even if there were noisy points nearby.

4.4. DROP4: More Carefully Filtering Noise

DROP4 is identical to DROP3 except that instead of blindly applying ENN, the noise-filtering pass removes each instance only if it is (1) misclassified by its k nearest neighbors, *and* (2) it does not hurt the classification of other instances. While DROP3 usually works well, it can in rare cases remove far too many instances in the noise-reduction pass (even all of them in one experiment). DROP4 avoids such problems and thus protects against especially poor generalization accuracy in such rare cases, at the expense of slightly higher storage requirements on average.

4.5. DROP5: Smoothing the Decision Boundary

DROP5 modifies DROP2 such that instances are considered for removal beginning with instances that are *nearest* to their nearest enemy, and proceeding outward. This serves as a noise-reduction pass, but will also cause most internal points to be removed as well. After this pass, the furthest-to-nearest pass as done by DROP2 is done repeatedly until no further improvement can be made.

A modified version of DROP5 was used in the *Reduced Probabilistic Neural Network* (RPNN) [Wilson & Martinez, 1997b], which is a *Radial Basis Function* (RBF) network used for classification. The RPNN used a pruning technique that included a conservative nearest-to-furthest noise-filtering pass followed by a more aggressive furthest-to-nearest node pruning pass.

4.6. Decremental Encoding Length

The *Decremental Encoding Length* (DEL) model is the same as DROP3, except that it uses the encoding length heuristic (as is used in *ELGrow* and *Explore* in Section 3.2) to decide in each case whether an instance can be removed. DEL starts with $S = T$, and begins with a noise-filtering pass in which each instance is removed if (a) it is misclassified by its k nearest neighbors, and (b) removing the instance does not increase the encoding length cost. The remaining instances are then sorted by the distance to their nearest enemy, and as long as any improvement is being made, the remaining instances are removed (starting with the instance furthest from its nearest enemy) if doing so does not increase the encoding length cost.

5. Experimental Results

Many of the reduction techniques surveyed in Section 3 and all of the techniques proposed in Section 4 were implemented and tested on 31 datasets from the Machine Learning Database Repository at the University of California, Irvine [Merz & Murphy, 1996]. Those included in these experiments are *CNN*, *SNN*, *ENN*, *RENN*, *All k-NN*, *IB2*, *IB3*, *ELGrow*, *Explore*, *DEL*, and *DROP1-5*.

These experiments were limited to those models that choose a subset S from the training set T to use for subsequent classification. Therefore, the methods that modify the instances themselves were not included, i.e., rule-based, prototype, and hyperrectangle-building methods. Similarly, *VSM* and *MCS* were excluded since they are part of more complicated systems. *RMHC* was excluded because it does not specify how many instances to retain, and its method is subsumed by *Explore*. Similarly, *RNN* and *Shrink (Subtractive)* are improved upon by *DROP2* and *DROP1*, respectively, and are thus not included for the sake of parsimony.

The basic k nearest neighbor (kNN) algorithm that retains 100% of the training set is also included for comparison.

All of the algorithms use $k = 3$, and in our experiments they all use the *HVDM* distance function. (Experiments were also done using a more traditional Euclidean distance metric with overlap metric for nominal attributes, but the average accuracy for all of the algorithms was higher using *HVDM*.)

5.1. Results

Ten-fold cross-validation was used for each experiment. For each dataset, each pruning technique was given a training set T consisting of 90% of the available data, from which it returned a subset S. The remaining 10% of the data was classified using only the instances in S, and the average accuracy over 10 such trials is reported for

Database	kNN	%	CNN	%	SNN	%	IB2	%	IB3	%	LED	%	Average	Avg%
Anneal	93.11	100	96.99	9.57	86.08	10.57	96.74	9.48	91.35	9.79	93.85	9.30	**92.32**	28.12
Australian	84.78	100	77.68	24.22	81.31	28.38	78.26	24.15	85.22	4.78	84.78	2.56	**82.35**	27.92
Breast Cancer(WI)	96.28	100	95.71	7.09	93.85	8.35	95.71	7.09	96.57	3.47	96.28	1.89	**94.56**	25.55
Bridges	66.09	100	61.18	49.48	61.37	52.52	62.18	48.64	64.73	28.83	64.27	35.64	**59.20**	37.74
Crx	83.62	100	79.42	24.15	81.59	27.52	79.42	24.15	86.09	4.28	83.62	3.08	**82.91**	27.85
Echocardiogram	94.82	100	85.18	14.72	48.75	26.29	85.18	14.72	72.86	11.57	93.39	6.91	**89.01**	30.31
Flag	61.34	100	53.63	50.29	53.63	51.66	53.11	49.95	49.47	34.14	56.18	45.88	**56.85**	39.50
Glass	73.83	100	68.14	38.53	64.39	42.63	66.77	39.25	62.14	33.80	69.59	38.42	**65.30**	38.95
Heart	81.48	100	70.00	26.17	77.04	33.78	70.00	26.17	80.00	13.58	78.89	4.73	**78.41**	30.68
Heart(Cleveland)	81.19	100	73.95	30.84	76.25	33.88	73.96	30.29	81.16	11.11	79.49	13.64	**79.00**	31.93
Heart(Hungarian)	79.55	100	70.40	28.87	75.84	34.01	73.87	27.44	79.20	9.90	77.18	12.28	**78.04**	30.13
Heart(Long Beach VA)	70.00	100	61.00	35.67	67.00	43.56	57.00	35.39	70.00	4.89	70.00	19.28	**70.26**	31.01
Heart(More)	73.78	100	69.69	33.21	72.22	43.64	69.69	33.21	76.31	9.36	75.15	16.81	**73.02**	30.99
Heart(Swiss)	92.69	100	91.09	11.38	92.69	15.90	90.26	11.38	93.46	3.70	92.69	4.25	**92.95**	26.05
Hepatitis	80.62	100	75.50	25.30	81.92	30.96	74.17	25.66	73.08	5.09	80.00	7.59	**78.69**	28.88
Horse Colic	57.84	100	59.90	35.66	64.47	48.65	60.24	35.36	66.75	8.49	67.73	21.82	**60.89**	27.38
Image Segmentation	93.10	100	90.00	16.61	77.38	13.02	89.52	16.93	92.14	16.01	91.90	11.11	**89.71**	30.43
Ionosphere	84.62	100	82.93	21.62	81.74	19.21	82.93	21.62	85.75	14.59	86.32	12.88	**83.99**	28.73
Iris	94.00	100	90.00	12.74	83.34	14.07	90.00	12.74	94.67	19.78	93.33	9.56	**92.27**	31.29
LED Creator+17	67.10	100	55.50	43.14	59.10	51.38	55.50	43.16	60.70	32.31	66.60	20.90	**66.21**	34.89
LED Creator	73.40	100	64.90	35.79	71.80	92.78	64.60	35.71	70.40	22.04	72.30	13.92	**70.79**	35.10
Liver (Bupa)	65.57	100	56.80	40.87	57.70	52.59	56.80	40.87	58.24	10.66	61.38	38.36	**60.33**	37.62
Pima Diabetes	73.56	100	65.76	36.89	67.97	42.95	65.76	36.89	69.78	10.97	71.61	12.64	**71.00**	33.04
Promoters	93.45	100	86.73	13.83	87.09	15.51	84.91	14.36	91.64	18.12	83.09	7.34	**88.64**	31.48
Sonar	87.55	100	74.12	32.85	79.81	28.26	80.88	33.87	69.38	12.02	83.29	29.86	**77.90**	37.66
Soybean (Large)	88.59	100	83.10	24.97	80.44	20.27	84.06	24.61	86.63	30.33	87.27	24.76	**84.81**	38.31
Vehicle	71.76	100	67.50	37.04	67.27	43.21	67.50	37.04	67.62	28.36	68.10	32.51	**66.80**	37.67
Voting	95.64	100	93.59	9.12	95.40	10.21	93.59	9.12	95.64	5.44	94.27	2.02	**94.44**	26.57
Vowel	96.57	100	86.72	30.05	78.56	19.97	87.48	29.71	89.57	36.60	93.17	36.15	**85.57**	47.48
Wine	94.93	100	92.65	14.30	96.05	14.23	92.65	14.30	91.50	16.60	94.38	9.05	**93.50**	30.91
Zoo	94.44	100	91.11	12.47	76.67	10.62	91.11	12.47	92.22	29.38	90.00	18.27	**91.05**	34.69
Average	**82.11**	100	**76.48**	26.69	**75.44**	31.63	**76.58**	26.64	**78.85**	16.13	**80.65**	16.88	**79.06**	32.54

Table 1(a) Results for CNN, SNN, IB2, IB3 and LED.

each reduction technique on each dataset in Table 1. Due to the size of Table 1, it is broken into three parts, but the overall average and the results for the *k*NN algorithm are included with each for comparison. The average percentages of instances in *T* that were included in *S* is also reported for each experiment under the column "%".

Several observations can be made from the results in this table. *CNN* and *IB2* both achieve almost identical results (less than 1% difference in both size and accuracy in most cases), due to the similarity of their algorithms. *SNN* had lower accuracy and higher storage requirements on average when compared to *CNN* and *IB2*, and the *SNN* algorithm is much more complex and significantly slower than the others as well. *IB3* was able to achieve higher accuracy and lower storage than any of these algorithms, with the only disadvantage being a learning algorithm that is somewhat more complex (though not much slower) than *CNN* or *IB2*.

As expected, *ENN*, *RENN* and *All k-NN* all retained over 75% of the instances, due to their retention of internal (non-border) instances. They all had fairly good accuracy, largely because they still had access to most of the original instances. In agreement with Tomek [1976], the *All k-NN* method achieved better reduction and higher accuracy than *RENN*, which in turn had higher reduction (though slightly lower accuracy) than *ENN*.

The encoding-length heuristic techniques had by far the best storage reduction of any of the algorithms. The *ELGrow* algorithm achieved the lowest average reduction (1.67%) but also suffered a significant drop in generalization accuracy when compared to the non-pruned system. However, the *Explore* method achieved better average accuracy with only a slight increase in storage over *ELGrow*, indicating that the

Database	kNN	%	DROP1	%	DROP2	%	DROP3	%	DROP4	%	DROP5	%	Average	Avg%
Anneal	93.11	100	87.70	5.05	95.61	8.08	94.11	8.65	94.36	11.67	95.24	9.93	**92.32**	28.12
Australian	84.78	100	64.49	2.30	83.62	7.28	83.91	5.96	84.78	7.99	83.91	9.18	**82.35**	27.92
Breast Cancer(WI)	96.28	100	77.52	1.14	95.86	3.13	96.14	3.58	96.28	4.05	95.71	4.07	**94.56**	25.55
Bridges	66.09	100	39.64	10.17	61.18	17.30	56.36	17.60	57.36	21.28	62.82	22.22	**59.20**	37.74
Crx	83.62	100	65.94	3.75	84.64	7.31	85.80	5.46	85.51	7.33	83.77	7.68	**82.91**	27.85
Echocardiogram	94.82	100	93.39	9.61	94.82	10.51	93.39	10.66	94.82	10.96	93.39	9.16	**89.01**	30.31
Flag	61.34	100	43.18	9.05	62.79	20.62	61.29	20.45	59.58	27.09	58.13	25.26	**56.85**	39.50
Glass	73.83	100	62.97	15.47	65.04	23.10	65.02	23.88	65.91	29.54	65.45	24.81	**65.30**	38.95
Heart	81.48	100	72.59	5.47	81.85	12.22	83.33	13.62	81.85	16.71	81.11	16.67	**78.41**	30.68
Heart(Cleveland)	81.19	100	70.91	6.09	79.55	11.92	80.84	12.76	78.19	15.26	79.84	15.37	**79.00**	31.93
Heart(Hungarian)	79.55	100	72.17	5.74	78.52	8.80	80.29	9.86	79.22	11.53	79.60	11.15	**78.04**	30.13
Heart(Long Beach VA)	70.00	100	69.00	4.39	70.00	11.83	73.50	4.50	74.00	11.72	73.00	14.94	**70.26**	31.01
Heart(More)	73.78	100	63.46	3.51	73.98	10.71	76.38	9.14	74.36	13.19	74.63	14.62	**73.02**	30.99
Heart(Swiss)	92.69	100	93.46	1.81	93.46	2.53	93.46	1.81	93.46	2.35	92.63	5.42	**92.95**	26.05
Hepatitis	80.62	100	72.38	4.66	80.75	10.54	81.87	7.81	78.75	9.75	83.29	9.39	**78.69**	28.88
Horse Colic	57.84	100	59.15	1.55	70.74	8.20	70.13	10.30	67.73	20.41	68.45	14.14	**60.89**	27.38
Image Segmentation	93.10	100	81.19	6.61	92.86	10.45	92.62	10.98	94.05	12.41	89.29	11.35	**89.71**	30.43
Ionosphere	84.62	100	79.77	3.23	86.60	7.79	87.75	7.06	86.90	10.60	86.90	9.78	**83.99**	28.73
Iris	94.00	100	84.67	8.59	94.67	14.22	95.33	14.81	95.33	14.89	94.00	12.15	**92.27**	31.29
LED Creator+17	67.10	100	61.40	9.94	69.20	12.98	70.40	12.66	69.50	16.37	69.80	14.96	**66.21**	34.89
LED Creator	73.40	100	68.30	10.05	71.80	11.85	71.70	11.93	71.90	13.71	72.00	12.33	**70.79**	35.10
Liver (Bupa)	65.57	100	58.24	10.92	67.77	24.77	60.84	24.99	62.60	32.56	65.50	31.08	**60.33**	37.62
Pima Diabetes	73.56	100	65.23	6.50	70.44	17.59	75.01	16.90	72.53	21.76	73.05	21.95	**71.00**	33.04
Promoters	93.45	100	87.00	6.39	84.91	13.63	86.82	16.67	86.82	16.67	87.00	12.58	**88.64**	31.48
Sonar	87.55	100	64.93	11.38	80.88	26.60	78.00	26.87	82.81	31.20	79.88	29.81	**77.90**	37.66
Soybean (Large)	88.59	100	77.20	19.51	86.60	22.77	84.97	25.26	86.29	28.41	83.73	25.44	**84.81**	38.31
Vehicle	71.76	100	59.91	12.07	67.37	21.49	65.85	23.00	67.03	27.88	70.22	26.71	**66.80**	37.67
Voting	95.64	100	93.11	2.91	94.50	4.90	95.87	5.11	95.87	5.36	95.86	7.13	**94.44**	26.57
Vowel	96.57	100	83.31	39.16	91.08	44.66	89.56	45.22	90.70	46.02	93.36	42.66	**85.57**	47.48
Wine	94.93	100	90.98	5.74	93.24	11.42	94.93	16.11	94.93	16.17	96.08	9.74	**93.50**	30.91
Zoo	94.44	100	88.89	18.02	88.89	15.80	90.00	20.00	91.11	21.60	95.56	17.16	**91.05**	34.69
Average	**82.11**	100	**72.65**	8.41	**81.07**	14.03	**81.14**	14.31	**81.11**	17.30	**81.39**	16.09	**79.06**	32.54

Table 1(b). Results for *DROP1-DROP5*.

random mutation hill climbing step was successful in finding a better subset *S* after the growing and pruning phases were complete. The *DEL* approach was able to achieve higher average accuracy than the *Explore* method, but its storage reduction was not as dramatic.

The ordered reduction techniques *DROP2-DROP5* all had generalization accuracy that was within 1% of the full *k*NN classifier. Their accuracy was higher than any of the other reduction methods. *DROP2* and *DROP3* both had storage requirements of about 14%, which is lower than any of the other methods except *ELGrow* and *Explore*. *DROP1* retained about half as many instances as the other ordered reduction techniques, but had the worst generalization accuracy of any of them, because it fails to use information provided by pruned instances in determining whether further instances should be pruned.

5.2. Effect of Noise

Since several of these algorithms are designed to be robust in the presence of noise, the same experiments were repeated with 10% noise artificially added to each dataset. This was done by randomly changing the output class of 10% of the instances in the training set to an incorrect value. The output class of the instances in the *test* set are not noisy, so the results indicate how well each model is able to predict the correct output even if some of its training data is faulty.

Table 2 shows the average accuracy and storage requirements over all 30 datasets for each algorithm, including the unpruned *k*NN algorithm.

Database	kNN	%	ENN	%	RENN	%	AllKNN	%	ELGrow	%	Explore	%	Average	Avg%
Anneal	93.11	100	89.73	91.99	89.48	90.70	89.98	92.43	88.35	0.70	91.11	0.75	**92.32**	28.12
Australian	84.78	100	84.49	86.49	84.20	84.80	86.09	78.07	83.62	0.32	85.80	0.32	**82.35**	27.92
Breast Cancer(WI)	96.28	100	97.00	96.80	96.86	96.61	97.00	94.58	89.86	0.32	96.71	0.32	**94.56**	25.55
Bridges	66.09	100	59.46	68.23	58.36	65.93	59.36	58.48	56.27	5.35	57.18	5.67	**59.20**	37.74
Crx	83.62	100	85.36	86.17	85.80	85.64	85.07	78.82	85.22	0.32	85.51	0.32	**82.91**	27.85
Echocardiogram	94.82	100	93.39	92.94	93.39	92.94	93.39	92.18	93.39	3.01	94.82	3.01	**89.01**	30.31
Flag	61.34	100	63.32	67.07	62.76	62.83	61.24	55.67	55.50	2.00	56.16	2.06	**56.85**	39.50
Glass	73.83	100	65.91	70.82	64.00	69.06	67.75	65.89	50.54	2.28	63.98	3.53	**65.30**	38.95
Heart	81.48	100	81.11	83.13	81.11	81.98	81.85	72.02	74.44	0.82	81.85	0.82	**78.41**	30.68
Heart(Cleveland)	81.19	100	82.49	83.46	82.16	82.51	81.51	72.72	81.52	0.73	82.15	0.73	**79.00**	31.93
Heart(Hungarian)	79.55	100	80.28	82.12	79.25	79.93	80.62	70.79	80.61	0.75	82.30	0.75	**78.04**	30.13
Heart(Long Beach VA)	70.00	100	74.00	75.44	74.00	72.72	74.00	62.28	72.50	0.84	74.50	1.11	**70.26**	31.01
Heart(More)	73.78	100	76.31	76.58	76.51	74.41	75.60	66.97	67.48	0.14	73.13	0.14	**73.02**	30.99
Heart(Swiss)	92.69	100	93.46	93.49	93.46	93.49	93.46	88.71	93.46	0.90	93.46	0.90	**92.95**	26.05
Hepatitis	80.62	100	81.25	83.73	80.58	82.80	81.33	75.20	76.67	1.00	78.67	1.29	**78.69**	28.88
Horse Colic	57.84	100	45.89	58.21	32.91	27.87	45.89	52.16	67.09	0.37	67.09	0.37	**60.89**	27.38
Image Segmentation	93.10	100	91.90	92.72	91.43	91.77	92.14	91.46	85.95	2.22	89.76	2.43	**89.71**	30.43
Ionosphere	84.62	100	84.04	84.24	84.04	82.27	84.05	82.18	73.77	0.63	80.89	0.63	**83.99**	28.73
Iris	94.00	100	95.33	94.74	95.33	94.67	95.33	93.78	88.67	2.30	92.67	2.30	**92.27**	31.29
LED Creator+17	67.10	100	71.00	71.42	70.90	70.00	70.90	58.98	71.20	1.66	72.20	1.40	**66.21**	34.89
LED Creator	73.40	100	72.10	73.88	72.00	72.86	71.80	72.07	70.40	1.53	72.10	1.52	**70.79**	35.10
Liver (Bupa)	65.57	100	61.12	68.15	58.77	63.13	60.24	52.34	56.74	0.55	57.65	0.64	**60.33**	37.62
Pima Diabetes	73.56	100	75.39	76.37	75.91	74.52	74.88	64.61	67.84	0.29	75.27	0.29	**71.00**	33.04
Promoters	93.45	100	93.45	96.33	93.45	96.33	93.45	95.07	88.82	2.10	91.36	2.10	**88.64**	31.48
Sonar	87.55	100	81.79	84.35	78.38	81.79	80.36	80.29	70.24	1.07	70.29	1.07	**77.90**	37.66
Soybean (Large)	88.59	100	86.61	89.90	85.97	87.41	86.62	88.24	82.70	7.35	85.92	7.78	**84.81**	38.31
Vehicle	71.76	100	69.52	73.81	69.05	69.75	70.21	64.74	58.15	2.25	60.76	2.47	**66.80**	37.67
Voting	95.64	100	95.41	95.84	95.41	95.81	95.41	94.35	88.99	0.51	94.25	0.51	**94.44**	26.57
Vowel	96.57	100	92.40	96.57	91.27	95.94	93.54	96.70	50.20	4.69	57.77	6.65	**85.57**	47.48
Wine	94.93	100	94.93	95.57	94.93	95.57	94.93	94.76	81.47	1.93	95.46	2.12	**93.50**	30.91
Zoo	94.44	100	91.11	92.96	91.11	92.59	93.33	94.07	94.44	7.90	95.56	8.40	**91.05**	34.69
Average	**82.11**	100	**80.95**	83.34	**80.09**	80.92	**81.01**	77.44	**75.68**	1.83	**79.24**	2.01	**79.06**	32.54

Table 1(c). Results for *ENN, RENN*, *All k-NN*, *ELGrow*, and *Explore*.

Algorithm	Clean	Size%	Noisy	Size%
kNN	**82.11**	100.00	**78.93**	100.00
CNN	**76.48**	26.69	**68.14**	38.29
SNN	**75.44**	31.63	**74.60**	48.60
IB2	**76.58**	26.64	**67.80**	38.27
IB3	**78.85**	16.13	**72.09**	18.85
DEL	**80.65**	16.88	**78.16**	8.57
DROP1	**72.65**	8.41	**71.24**	8.00
DROP2	**81.07**	14.03	**79.99**	14.75
DROP3	**81.14**	14.31	**80.00**	11.49
DROP4	**81.11**	17.30	**79.57**	14.74
DROP5	**81.39**	16.09	**79.95**	15.52
ENN	**80.95**	83.34	**80.19**	74.91
RENN	**80.09**	80.92	**79.65**	72.53
AllKNN	**81.01**	77.44	**79.98**	64.58
ELGrow	**75.68**	1.83	**73.67**	1.88
Explore	**79.24**	2.01	**77.96**	2.03
Average	**79.06**	32.54	**76.36**	32.83

Table 2. Average accuracy and storage requirements in the presence of 10% noise.

As can be seen from Table 2, the accuracy for the *k*NN algorithm dropped just over 3% on average. Note that some of the effect of noise is already handled by the use of $k = 3$ in these experiments. Otherwise the drop in accuracy would be more on the order of 8% (i.e., 10% of the 82% already classified correctly).

As expected, CNN and IB2 increased storage and suffered large reductions in accuracy in the presence of noise. SNN dropped only slightly in accuracy when noise was added, but it retained almost half of the instances in the training set due to its strict (and noise intolerant) requirements as to which instances must be in S.

In agreement with Aha's results [1992], IB3 had higher accuracy and lower storage requirements in the presence of noise than IB2, though it still suffered a dramatic decrease in accuracy (and a slight increase in storage) when compared to its performance in the noise-free case. In our experiments we found that when the number of instances in the training set was small, IB3 would occasionally end up with an *empty* subset S, because none of the instances gets enough statistical strength to be *acceptable*. This problem worsens in the presence of noise, and thus more training data (or a modification of the algorithm) is required to handle small, noisy datasets.

DROP1 did not fall much in accuracy, but its accuracy was already poor to begin with. However, all of the other DROP methods (DROP2-5) all achieved accuracy *higher* than the *k*NN method, while using less than one-sixth of the original instances. DROP3 had the highest accuracy of the DROP methods, and had the lowest storage of the accurate ones (DROP2-5), using less than 12% of the original instances.

The *ENN, RENN*, and *All k-NN* methods also achieved higher accuracy than *k*NN, since they were designed specifically for noise filtering. They also required about 10% less storage than in the noise-free case, probably because they were throwing most of the noisy instances (as well as a few good instances that were made to appear noisy due to the added noise).

The encoding-length heuristic methods all dropped about 2% in accuracy when noise was added leaving them closer to—but still below—the *k*NN method in terms of accuracy. *ELGrow* had fairly poor accuracy compared to the others, but *Explore* was within 1% of the *k*NN method in terms of accuracy while using only about 2% of the instances for storage.

6. Conclusions and Future Research Directions

Many techniques have been proposed to reduce the number of instances used for classification in distance-based learning algorithms. In experiments on 31 datasets, the results make possible the division of the tested algorithms into several groups. The first group consists of algorithms which had low generalization accuracy and are thus mostly of historical significance. This group includes *CNN*, *SNN*, *IB2* (which led to the development of *IB3*) and *DROP1* (which led to the more successful *DROP* algorithms). These had low generalization even before noise was introduced, and dropped further when it was. Of this group, only *DROP1* kept less than 25% of the instances on average, so the storage reduction did not make up for the lack of accuracy.

The second group consists of the three similar noise-filtering algorithms: *ENN, RENN*, and *All k-NN*. These had high accuracy but also kept most of the instances. In the noise-free environment, they achieved slightly lower accuracy than *k*NN, but when noise was added, their accuracy was higher than *k*NN, indicating that they are successful in the situation for which they were designed. These algorithms are useful when there noise is expected in the data and when it is reasonable to retain most of the data. Of this group, *All k-NN* had the highest accuracy and lowest storage requirements in the presence of noise.

The third group consists of two algorithms, *ELGrow* and *Explore*, that were able to achieve reasonably good accuracy with only about 2% of the data. *ELGrow* had the lowest storage (about 1.8%) but its accuracy was somewhat poor, especially in the noisy domain. The *Explore* method had fairly good accuracy, especially in the noisy arena, though it was not quite as accurate as the *DROP* methods. However, its aggressive storage reduction would make this trade-off acceptable in many cases.

The final group consists of algorithms which had high accuracy and reasonably good storage reduction. These include *IB3*, *DROP2-5* and *DEL*. *IB3* was designed to overcome the noise-sensitivity of *IB2*, and in our experiments it had better accuracy and storage reduction than *IB2*, especially in the noisy case. The algorithms *DROP2-DROP5* had even higher accuracy than *IB3* and on average improved in terms of storage reduction as well. They all had an accuracy within about 1% of *k*NN on the original data, and were about 1% higher when noise was added, with storage ranging from 11% to 18%. *DEL* had slightly lower accuracy, but also had lower storage in the noisy domain.

This paper has reviewed much of the work done in the area of reducing storage requirements in exemplar-based learning systems. The effect of noise on many of the algorithms has also been observed on a collection of datasets. Other factors that influence the successfulness of each algorithm must still be identified. Current research is seeking to determine under what conditions each of these algorithms is successful so that an appropriate algorithm can be chosen when needed. Current research is also focused on combining the reduction techniques proposed here with various weighting techniques in order to develop learning systems that can more dynamically adapt to problems of interest.

Appendix

The on-line appendix contains the cost function for the encoding-length methods, as well as the complete source code used for these experiments. It also contains a copy of the datasets used in the experiments presented in this paper. It is available at

ftp://axon.cs.byu.edu/pub/randy/ml/appendix/

References

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

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.

Batchelor, Bruce G., (1978). *Pattern Recognition: Ideas in Practice*. New York: Plenum Press, pp. 71-72.

Biberman, Yoram, (1994). A Context Similarity Measure. *In Proceedings of the European Conference on Machine Learning (ECML-94)*. Catalina, Italy: Springer Verlag, pp. 49-63.

Brodley, Carla E., (1993). “Addressing the Selective Superiority Problem: Automatic Algorithm/Model Class Selection,” *Proceedings of the Tenth International Machine Learning Conference,* Amherst, MA, pp. 17-24.

Broomhead, D. S., and D. Lowe (1988). Multi-variable functional interpolation and adaptive networks. *Complex Systems*, vol. 2, pp. 321-355.

Cameron-Jones, R. M., (1995). Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing. In *Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence*, pp. 99-106.

Carpenter, Gail A., and Stephen Grossberg, (1987). A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine. *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115.

Chang, Chin-Liang, (1974). “Finding Prototypes for Nearest Neighbor Classifiers,” *IEEE Transactions on Computers*, vol. 23, no. 11, November 1974, pp. 1179-1184.

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.

Dietterich, Thomas G., (1989). Limitations on Inductive Learning. In *Proceedings of the Sixth International Conference on Machine Learning*. San Mateo, CA: Morgan Kaufmann, pp. 124-128.

Diday, Edwin, (1974). Recent Progress in Distance and Similarity Measures in Pattern Recognition. *Second International Joint Conference on Pattern Recognition*, pp. 534-539.

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).*

Dudani, Sahibsingh A., (1976). “The Distance-Weighted *k*-Nearest-Neighbor Rule,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, no. 4, April 1976, pp. 325-327.

Gates, G. W. (1972). “The Reduced Nearest Neighbor Rule,” *IEEE Transactions on Information Theory*, vol. IT-18, no. 3, pp. 431-433.

Hart, P. E., (1968). “The Condensed Nearest Neighbor Rule,” Institute of Electrical and Electronics Engineers Transactions on Information Theory, vol. 14, pp. 515-516.

Hecht-Nielsen, R., (1987). Counterpropagation Networks. *Applied Optics*, vol. 26, no. 23, pp. 4979-4984.

Kibler, D., and David W. Aha, (1987). “Learning representative exemplars of concepts: An initial case study.” *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, CA: Morgan Kaufmann, pp. 24-30.

Lowe, David G., (1995). “Similarity Metric Learning for a Variable-Kernel Classifier,” *Neural Computation*., vol. 7, no. 1, pp. 72-85.

Merz, C. J., and P. M. Murphy, (1996). *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: http://www.ics.uci.edu/ ~mlearn/ MLRepository.html.

Michalski, Ryszard S., Robert E. Stepp, and Edwin Diday, (1981). A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts. *Progress in Pattern Recognition*, vol. 1, Laveen N. Kanal and Azriel Rosenfeld (Eds.). New York: North-Holland, pp. 33-56.

Nadler, Morton, and Eric P. Smith, (1993). *Pattern Recognition Engineering*. New York: Wiley, pp. 293-294.

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.

Papadimitriou, C. H., and Steiglitz, K., (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ.

Renals, Steve, and Richard Rohwer, (1989). Phoneme Classification Experiments Using Radial Basis Functions. *In Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'89)*, vol. 1, pp. 461-467.

Ritter, G. L., H. B. Woodruff, S. R. Lowry, and T. L. Isenhour, (1975). "An Algorithm for a Selective Nearest Neighbor Decision Rule," *IEEE Transactions on Information Theory*, vol. 21, no. 6, November 1975, pp. 665-669.

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

Salzberg, Steven, (1991). "A Nearest Hyperrectangle Learning Method," *Machine Learning*, vol. 6, pp. 277-309.

Schaffer, Cullen, (1994). A Conservation Law for Generalization Performance. *In Proceedings of the Eleventh International Conference on Machine Learning (ML'94)*, Morgan Kaufmann, 1994.

Skalak, D. B., (1994). Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning (ML94)*. Morgan Kaufmann, pp. 293-301.

Specht, Donald F., (1992). "Enhancements to Probabilistic Neural Networks," in *Proceedings International Joint Conference on Neural Networks (IJCNN '92)*, vol. 1, pp. 761-768.

Sproull, Robert F., (1991). Refinements to Nearest-Neighbor Searching in *k*-Dimensional Trees. *Algorithmica*, vol. 6, pp. 579-589.

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

Tomek, Ivan, (1976). "An Experiment with the Edited Nearest-Neighbor Rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 6, June 1976, pp. 448-452.

Tversky, Amos, (1977). Features of Similarity. *Psychological Review*, vol. 84, no. 4, pp. 327-352.

Wasserman, Philip D., (1993). *Advanced Methods in Neural Computing*. New York, NY: Van Nostrand Reinhold, pp. 147-176.

Watson, I., and F. Marir, (1994). "Case-Based Reasoning: A Review," *The Knowledge Engineering Review*, vol. 9, no. 4.

Wettschereck, Dietrich, (1994). "A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm", *To appear in the Proceedings of the 7th European Conference on Machine Learning*.

Wettschereck, Dietrich, and Thomas G. Dietterich, (1995). "An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms," *Machine Learning*, vol. 19, no. 1, pp. 5-28.

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.

Wilson, D. Randall, and Tony R. Martinez, (1996). "Heterogeneous Radial Basis Functions," *Proceedings of the International Conference on Neural Networks (ICNN'96)*, vol. 2, pp. 1263-1267, June 1996.

Wilson, D. Randall, and Tony R. Martinez, (1997a). "Improved Heterogeneous Distance Functions," *Journal of Artificial Intelligence Research (JAIR)*, vol. 6, no. 1, pp. 1-34.

Wilson, D. Randall, and Tony R. Martinez, (1997b). "Improved Center Point Selection for Radial Basis Function Networks," In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97)*.

Wilson, D. Randall, and Tony R. Martinez, (1997c). "Instance Pruning Techniques," To appear in Fisher, D., ed., *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, Morgan Kaufmann Publishers, San Francisco, CA.

Wilson, Dennis L., (1972). "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, pp. 408-421.

Wolpert, David H., (1993). On Overfitting Avoidance as Bias. Technical Report SFI TR 92-03-5001. Santa Fe, NM: The Santa Fe Institute.

Zhang, Jianping, (1992). "Selecting Typical Instances in Instance-Based Learning," *Proceedings of the Ninth International Conference on Machine Learning*.

Part IV

Attribute and Vote Weighting

"Dieting doesn't make you live longer. It just seems longer."

Instance-based learning algorithms can be sensitive to irrelevant attributes, so it is important to have some way of identifying and removing the effect of such attributes. Chapter 8 presents an instance-based learning system that uses a genetic algorithm to find attribute weights that maximize leave-one-out classification accuracy on the training set. This chapter was published in

Wilson, D. Randall, and Tony R. Martinez, (1996). "Instance-Based Learning with Genetically Derived Attribute Weights," *International Conference on Artificial Intelligence, Expert Systems and Neural Networks (AIE'96),* pp. 11-14.

Another criticism of the nearest neighbor algorithm is that once it stores the training set, it has no ability to adjust the decision boundary. The attribute weights derived in Chapter 8 provide one way to alter the concept description.

Chapter 9 presents another technique called the *Fuzzy Instance-Based Learning* (FIBL) algorithm, which uses distance-weighted voting to provide more flexibility in how the decision boundaries can be constructed and shows how a combination of cross-validation and confidence can be used to tune parameters with more precision than using either technique alone. This chapter has been submitted to

Wilson, D. Randall, and Tony R. Martinez, (1997). "Distance-Weighting and Confidence in Instance-Based Learning," submitted to *Computational Intelligence*.

Chapter 8

Instance-Based Learning with Genetically Derived Attribute Weights

“The gene pool could use a little chlorine.”

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

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 [Cover & Hart, 1967; Dasarathy, 1991; Aha, 1991; Aha, 1992; Cost & Salzberg, 1993; Domingos, 1995; Stanfill & Waltz, 1986] 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 [Cover & Hart, 1967; Dasarathy, 1991; Aha, 1991; Aha, 1992; Cost & Salzberg, 1993; Domingos, 1995; Stanfill & Waltz, 1986] 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

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

2.1. 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.

2.2. 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 Giraud-Carrier & Martinez [1995]. The distance between two values a and b of a given attribute i is given as

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

where *overlap* and *difference* are defined as

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

$$difference_i(a,b) = \frac{|a-b|}{range(i)} \tag{4}$$

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

$$range(i) = max(i) - min(i). \tag{5}$$

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

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

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.

2.3. 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.

2.4. 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 [Wilson, 1994]. 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 [1995] 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 [Spears et al., 1993] 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.

3.1. 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.

3.2. 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.

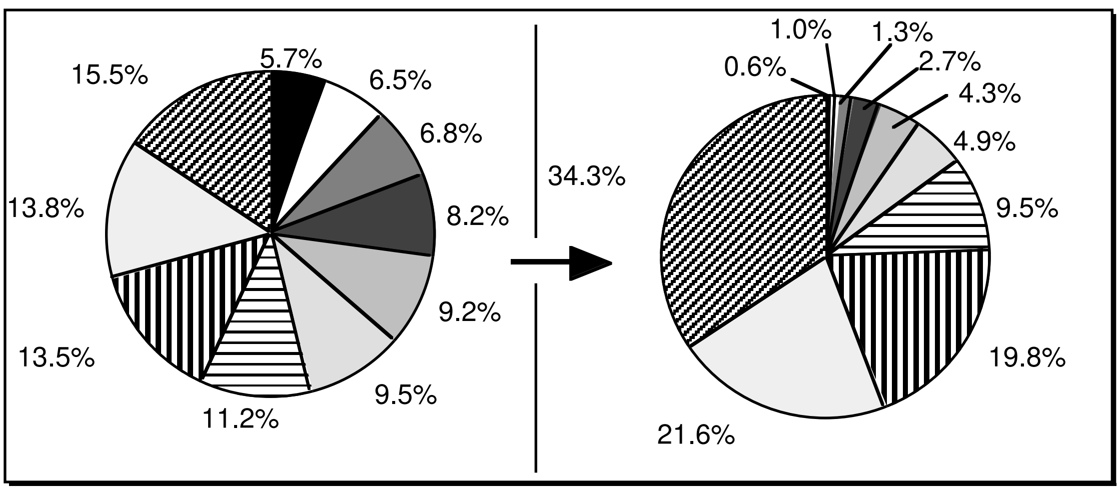


Figure 3. Graphical representation of parent selection probabilities.

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.

3.3. 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* [Papadimitriou & Bentley, 1980] 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 [Murphy & Aha, 1993]. 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.

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 5. Experimental results on data sets without irrelevant or redundant attributes.

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 5. 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.

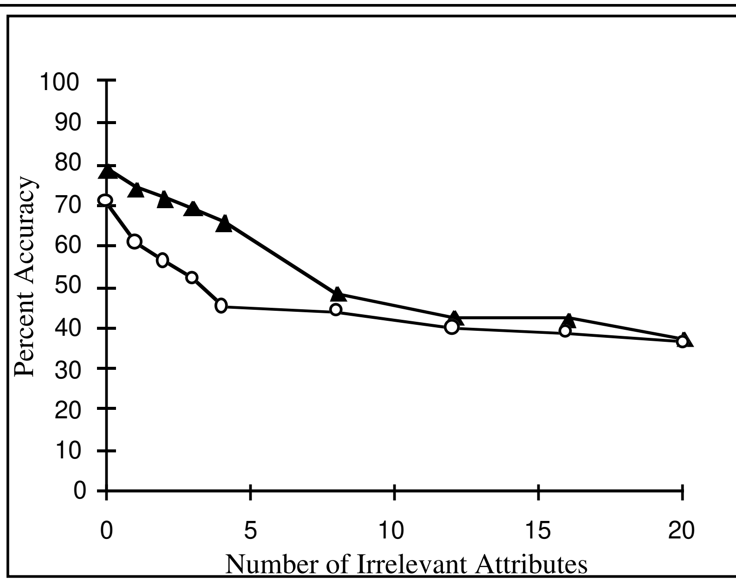
4.1. Testing Irrelevant Attributes

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 6 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 6, 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

GIBL
Non-Weighted

Figure 6. Glass data with irrelevant attributes.

4.2. 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 7.

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.

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 7. Glass data with redundant attributes.

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 non-weighted instance-based system. Current research includes the following:

- 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.

References

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.

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

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

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.

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

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).*

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.

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.

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.

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.

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

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.

Wilson, D. Randall, (1994). *Prototype Styles of Generalization*, Master’s Thesis, Brigham Young University.

Chapter 9

Distance-Weighting and Confidence in Instance-Based Learning

"I'm not afraid of heights. I'm afraid of widths."
—Steven Wright.

Submitted to *Computational Intelligence*, 1997.

Abstract

Many extensions have been proposed to help instance-based learning algorithms perform better on a wide variety of real-world applications. However, it is not trivial to decide what parameters or options to use when applying an instance-based learning algorithm to a particular problem. Traditionally, cross-validation has been used to choose some parameters such as k in a k-nearest neighbor classifier. This paper points out why cross validation often does not provide enough information to allow for fine-tuning the classifier, and how confidence levels can be used to break ties that are all too common when cross-validation is used. It proposes the *Fuzzy Instance Based Learning* (FIBL) algorithm that uses distance-weighted voting with parameters set via a combination of cross-validation and confidence levels. In experiments on 31 datasets, FIBL had higher average generalization accuracy than using majority voting or using cross-validation alone to determine parameters.

1. Introduction

Instance-Based Learning (IBL) [Aha, Kibler & Albert, 1991; Aha, 1992] is a paradigm of learning in which algorithms typically store some or all of the n available training examples (*instances*) from a *training set*, T, during learning. Each instance has an *input vector* $\boldsymbol{x}$, and an *output class* c. During *generalization*, these systems use a distance function to determine how close a new input vector $\boldsymbol{y}$ is to each stored instance and use the nearest instance or instances to predict the output class of $\boldsymbol{y}$ (i.e., to *classify* $\boldsymbol{y}$). Some of the earliest instance-based learning algorithms are referred to as nearest neighbor techniques [Cover & Hart, 1967; Dasarathy, 1991]. Related paradigms include *memory-based reasoning* methods [Stanfill & Waltz, 1986; Cost & Salzberg, 1993; Rachlin et al., 1994], *exemplar-based generalization* [Salzberg, 1991; Wettschereck & Dietterich, 1995], and *case-based reasoning* (CBR) [Watson & Marir, 1994]. Such algorithms have had much success on a wide variety of *applications* (real-world classification tasks).

The original nearest neighbor rule [Cover & Hart, 1967] uses the nearest instance in T to a new input vector $\boldsymbol{y}$ to choose an output class. Cover & Hart [1967] also mentioned the possibility of using the most commonly occurring class among the nearest k neighbors to classify $\boldsymbol{y}$ instead of just using the single nearest neighbor. The use of $k > 1$ can lessen the effect of noise in the system, but it also introduces a parameter to the system which must be chosen.

Dudani [1976] proposed a distance-weighted nearest neighbor algorithm in which neighbors nearer to the input vector get more weight. This can reduce the sensitivity of the system to the parameter k, but a suitable value for k must still be found.

One of the most popular methods of choosing k or other parameters is through the use of *cross-validation* (CV) [Schaffer, 1993; Moore & Lee, 1993; Kohavi, 1995]. In *J-fold* cross-validation, the training set T is divided into J partitions, $T_1...T_J$, and the instances in each of the partitions are classified by the instances in the remaining partitions using the proposed parameter value. The average accuracy of these J trials is used to estimate what the generalization accuracy would be if the parameter value was used. The parameter value that yields the highest estimated accuracy is then chosen.

When J is equal to the number of instances in T, the result is *leave-one-out* cross-validation (LCV), in which each instance i is classified by the instances in T-i, so that almost all of the data is available for each classification attempt. LCV is often described as being desirable but computationally expensive [Moore & Lee, 1993]. However, in our implementation (as described in Section 4), LCV is performed efficiently during the learning stage in a way that actually makes it faster than using larger partitions.

One problem with using CV or LCV to fine-tune a learning system (*e.g.*, when deciding whether to use $k = 1$ or $k = 3$, or when deciding what weight to give to an input attribute) is that it can yield only a fixed number of discrete accuracy estimates. Given n instances in a training set, CV will yield an accuracy estimation of 0 or 1 for each instance, yielding an average estimate that is in the range 0..1, but only in increments of $1/n$. This is equivalent in its usefulness to receiving an integer r in the range $0..n$ indicating *how many* of the instances are classified correctly using the current parameter settings.

Usually a change in any given parameter will not change r by more than a small value, so that the change in r given a change in parameter is usually a small integer such as -3...3. Unfortunately, changes in parameters often have *no* effect on r, in which case CV does not provide helpful information as to which alternative to choose. This problem occurs quite frequently in some systems and limits the extent to which CV can be used to fine-tune an instance-based learning model.

This paper presents a new instance-based learning system that uses distance-weighted voting to produce confidence levels and to make decisions when LCV is unable to do so. The combination of Cross-Validation and Confidence (CVC) is used to decide on the value of k, how fast and in what shape the voting weight drops off with distance and two other parameters as well.

Section 2 describes the *Fuzzy Instance-Based Learning* (FIBL) algorithm, and introduces the various decisions that have to be made before using the system for classification. Section 3 discusses how these decisions can be made using LCV combined with confidence estimations. It also mentions how confidence values can also be used during classification to indicate how confident the system is about its decision. Section 4 presents experimental results which show that FIBL is able to achieve higher average generalization accuracy than similar systems that either use default values or use CV alone to make decisions. Section 5 provides conclusions and future research directions.

2. Fuzzy Instance-Based Learning (FIBL) Algorithm

In the basic k-NN algorithm, the k nearest neighbors of an input vector each get 1 vote for their respective classes and the class that receives the most votes is chosen as the output class for the new input vector. The *Fuzzy Instance-Based Learning* (FIBL) algorithm, on the other hand, uses principles from *fuzzy logic* [Zadeh, 1965] to find a *class membership* of the input vector for each class. The class with the highest class membership is used as the output class, which is similar to using the majority class in the k-NN scheme. However, the class membership can also be used to indicate the confidence of classification and can be used to help tune parameters, as discussed in Section 3.

The *Fuzzy Instance-Based Learning* (FIBL) algorithm uses distance-weighted k-nearest neighbor voting, similar to that done by Dudani [1976], which allows decision boundaries to be fine-tuned with more precision than is allowed with simple majority voting. It also uses a *heterogeneous* distance function that is appropriate for domains with nominal attributes, linear attributes, or both. This section elaborates on these two topics.

2.1. Vote Weighting

Let $\boldsymbol{y}$ be the input vector to be classified and let $n_1 ... n_k$ be the k nearest neighbors of $\boldsymbol{y}$ in T. Let D_j be the distance from $\boldsymbol{y}$ to the jth neighbor using some distance function D (such as the HVDM distance function presented in Section 2.2).

In the FIBL algorithm, the voting weight of each of the k nearest neighbors depends on its distance from the input vector $\boldsymbol{y}$. The weight is 1 when the distance is 0 and decreases as the distance grows larger. The way in which the weight decreases as the distance grows depends on which *shape function* is used. The shapes used in FIBL are: *majority, linear, gaussian,* and *exponential.*

In majority voting, all k neighbors get an equal vote of 1. With the other three shapes, the voting weight of a neighbor n_j is 1 when the distance to n_j is 0 and drops to the value of a parameter w_k at a distance D_k, where D_k is the distance to the kth nearest neighbor.

Given w_k and D_k, the amount of voting weight w_j for the jth neighbor that is a distance D_j from the input vector for each shape is given in Equations 1-4.

$$\text{(a) Majority: } w_j = 1 \tag{1}$$

$$\text{(b) Linear: } w_j = w_k + \frac{(1 - w_k)(D_k - D_j)}{D_k} \tag{2}$$

$$\text{(c) Gaussian: } w_j = w_k^{D_j^2 / D_k^2} \tag{3}$$

$$\text{(d) Exponential: } w_j = w_k^{D_j / D_k} \tag{4}$$

Note that the majority voting scheme does not require the w_k parameter. Also note that if $k = 1$ or $w_k = 1$, then all four of these schemes are equivalent. As D_k approaches 0, the weight in Equations (2)-(4) all approach 1. Therefore, if the distance D_k is equal to 0, then a weight of 1 is used for consistency and to avoid dividing by 0. These four shapes are illustrated in Figure 1.

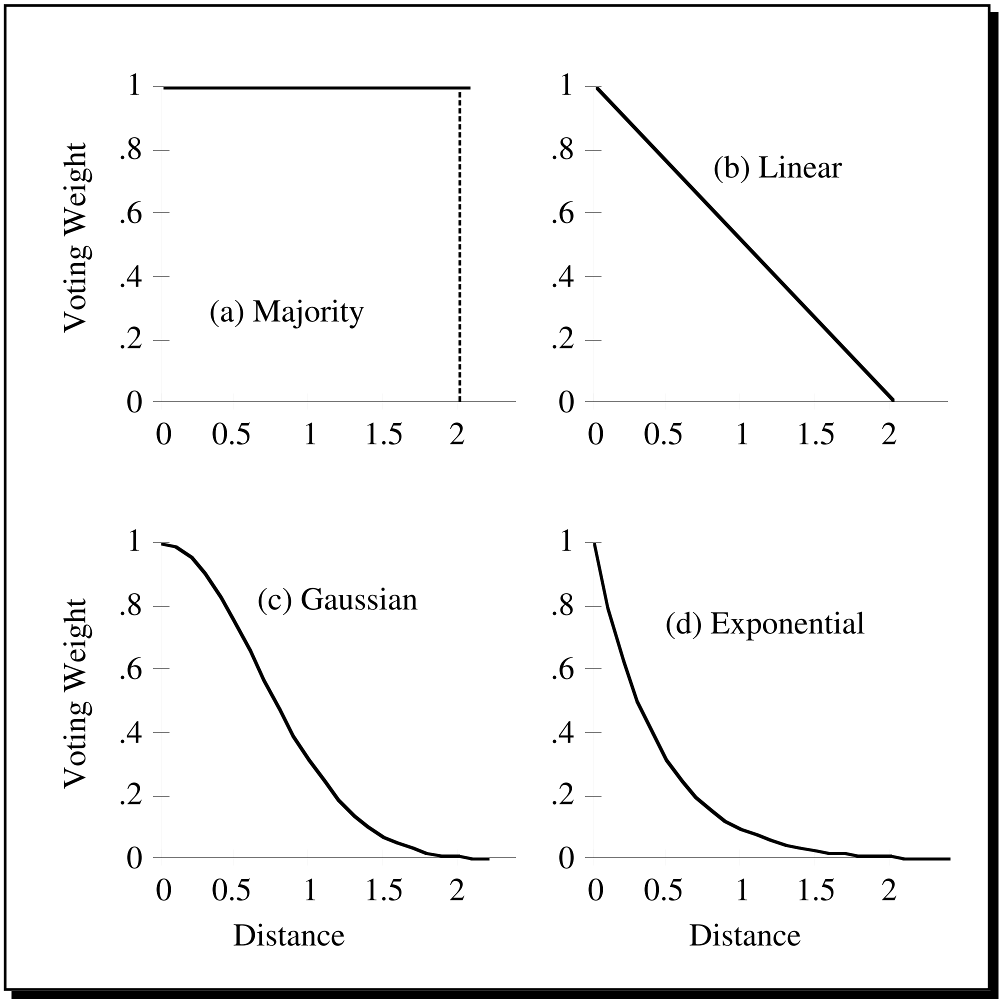


Figure 1. Distance-weighting shapes, shown with $D_k = 2.0$ and $w_k = .01$.

Sometimes it is preferable to use the *average* distance of the k nearest neighbors instead of the distance to the kth neighbor to determine how fast voting weight should drop off. This can be done by computing what the distance D'_k of the kth nearest neighbor would be if the neighbors were distributed evenly. This can be done by setting D'_k to

$$D'_k = \frac{2 \cdot \sum_{i=1}^{k} D_i}{k+1} \tag{5}$$

and using D'_k in place of D_k in Equations (2)-(4). When $k = 1$, Equation 5 yields $D'_k = 2D_k/2 = D_k$, as desired. When $k > 1$, this method can be more robust in the presence of changes in the system such as changing parameters or the removal of instances from the classifier.

2.2. Heterogeneous Distance Function

The distance function is critical to the success of an instance-based algorithm. A variety of distance functions are available, including the Minkowsky [Batchelor, 1978], Mahalanobis [Nadler & Smith, 1993], Camberra, Chebychev, Quadratic, Correlation and Chi-square distance metrics [Michalski, Stepp & Diday, 1981; Diday, 1974]; the Context-Similarity measure [Biberman, 1994]; the Contrast Model [Tversky, 1977]; hyperrectangle distance functions [Salzberg, 1991; Domingos, 1995] and others.

Although there have been many distance functions proposed, by far the most commonly used is the Euclidean Distance function (which is a special case of the Minkowsky metric). However, many applications have *nominal* (discrete, unordered) attributes, and Euclidean distance and other linear metrics are not appropriate for such attributes. For example, given an attribute *color* with values *red, green, blue,* and *brown*, there is no linear relationship in the arbitrarily ordered values that would indicate that *red* should be closer to *green* than to *brown*.

It is possible to use the *overlap* metric for nominal attributes, for which the distance is 0 if the attribute values are equal and 1 if they are different [Giraud-Carrier & Martinez, 1995; Aha, Kibler & Albert, 1991; Aha, 1992]. However, the overlap metric fails to give any indication of *how* different two unequal values are.

The *Value Difference Metric* (VDM) [Stanfill & Waltz, 1986; Cost & Salzberg, 1993; Rachlin et al., 1994] is able to return a real-valued distance between each pair of values for nominal attributes based on statistics gathered from the training set. On the other hand, the VDM does not directly handle linear attributes but instead requires *discretization* [Lebowitz, 1985], which can lead to inferior generalization accuracy [Ventura & Martinez, 1995].

We previously introduced several new heterogeneous distance functions that substantially improved average generalization accuracy on a collection of 48 different datasets [Wilson & Martinez, 1997]. FIBL uses one of the most successful functions, the *Heterogeneous Value Difference Metric* (HVDM). This distance function is briefly defined below, and more details on this distance function are available in [Wilson & Martinez, 1997].

The HVDM distance function defines the distance between two input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ as

$$HVDM(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{a=1}^{m} d_a(x_a, y_a)^2} \qquad (6)$$

where m is the number of attributes. The function $d_a(x,y)$ returns a distance between the two values x and y for attribute a and is defined as

$$d_a(x,y) = \begin{cases} 1 & \text{if } x \text{ or } y \text{ is unknown; otherwise...} \\ vdm_a(x,y) & \text{if } a \text{ is nominal} \\ \dfrac{|x-y|}{4\sigma_a} & \text{if } a \text{ is numeric} \end{cases} \qquad (7)$$

where σ_a is the sample standard deviation of the numeric values occurring for attribute a. Since 95% of the values in a normal distribution fall within two standard deviations of the mean, the difference between numeric values is divided by four standard deviations to scale each value into a range that is usually of width 1. The function $vdm_a(x,y)$ returns the distance between two nominal attribute values x and y as follows.

$$vdm_a(x,y) = \sum_{c=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2 \tag{8}$$

In Equation 8, $N_{a,x}$ is the number of times attribute a had value x in the training set; $N_{a,x,c}$ is the number of times attribute a had value x and the output class was c; and C is the number of output classes. Using this distance measure, two nominal attribute values are considered to be closer if they have more similar classifications, regardless of the order of the values.

The HVDM distance function is used to find distances in the FIBL algorithm, which in turn are used to weight voting and thus influence confidence, as discussed further in Section 3.

3. Cross-Validation and Confidence (CVC)

In Section 2, the FIBL system was introduced, and several parameters were mentioned without specifying how they are set. Specifically, for a given application, the following parameters need to be set:

- k, the number of neighbors that vote on the class of a new input vector,
- *shape*, the shape of the distance-weighted voting function,
- *avgk*, the flag determining whether to use the average distance to the k nearest neighbors rather than the kth distance.
- w_k, the weight of the kth neighbor (except in majority voting).

When faced with a choice between two or more sets of parameter values, some method is needed for deciding which is most likely to yield the best generalization. This section describes how these parameters are set automatically, using a combination of leave-one-out cross-validation and confidence levels.

3.1. Cross-Validation

With leave-one-out cross-validation (LCV), the generalization accuracy of a model is estimated from the average accuracy attained when classifying each instance i using all the instances in T except i itself. For each instance, the accuracy is 1 if the instance is classified correctly and 0 if it is misclassified. Thus the average LCV accuracy is r / n, where r is the number classified correctly and n is the number of instances in T. Since r is an integer from 0 to n, there are only $n + 1$ accuracy values possible with this measure, and often two different sets of parameter values will yield the same accuracy because they will classify the same number of instances correctly. This makes it difficult to tell which parameter values are better than another.

3.2. Confidence

An alternative method for estimating generalization accuracy is to use the *confidence* with which each instance is classified. The average confidence over all n instances in the training set can then be used to estimate which set of parameter values will yield better generalization. The confidence for each instance is

$$conf = \frac{votes_{correct}}{\sum_{c=1}^{C} votes_c} \tag{9}$$

where $votes_{correct}$ is the sum of weighted votes received for the correct class and $votes_c$ is the sum of weighted votes received for class c. In terms of fuzzy logic [Zadeh, 1965], the confidence of each instance can also be thought of as its class membership in the correct class.

When majority voting is used, $votes_c$ is simply a count of how many of the k nearest neighbors were of class c, since the weights are all equal to 1. In this case, the confidence will be an integer in the range $0..k$, divided by k, and thus there will be only $k + 1$ possible confidence values for each instance. This means that there will be $(k + 1)(n + 1)$ possible accuracy estimates using confidence instead of $n + 1$ as with LCV, but it is still possible for small changes in parameters to yield no difference in average confidence.

When distance-weighted voting is used, however, each vote is weighted according to its distance and the current set of parameter values, and $votes_c$ is the sum of the weighted votes for each class. Even a small change in the parameters will affect how much weight each neighbor gets and thus will affect the average confidence.

After learning is complete, the confidence can be used to indicate how confident the classifier is in its generalized output. In this case the confidence is the same as defined in Equation 9, except that $votes_{correct}$ is replaced with $votes_{out}$, which is the amount of voting weight received by the class that is chosen to be the output class by the classifier. This is also equal to the maximum number of votes (or maximum sum of voting weights) received by any class, since the majority class is chosen as the output.

3.3. Cross-Validation and Confidence (CVC)

Average confidence has the attractive feature that it provides a continuously valued metric for evaluating a set of parameter values. However, it also has drawbacks that make it inappropriate for direct use on the parameters in FIBL. Average confidence is increased whenever the ratio of votes for the correct class to total votes is increased. Thus, this metric strongly favors $k = 1$ and $w_k = 0$, regardless of their effect on classification, since these settings give the nearest neighbor more relative weight, and the nearest neighbor is of the same class more often than other neighbors. This metric also tends to favor exponential weighting since it drops voting weight more quickly than the other shapes.

Therefore, using confidence as the sole means of deciding between parameter settings will favor any settings that weight nearer neighbors more heavily, even if accuracy is degraded by doing so.

In order to avoid this problem, FIBL combines cross-validation and confidence into a single metric called *CVC*. Using CVC, the accuracy estimate cvc_i of a single instance i is

$$cvc_i = \frac{n \cdot cv + conf}{n+1} \tag{10}$$

where n is the number of instances in the training set; *conf* is as defined in Equation 9; and *cv* is 1 if instance i is classified correctly by its neighbors in T, or 0 otherwise.

This metric weights the cross-validation accuracy more heavily than the confidence by a factor of n. This technique is equivalent to using *numCorrect* + *avgConf* to make decisions, where *numCorrect* is the number of instances correctly classified by their neighbors and is an integer in the range 0..n, and *avgConf* is the average *conf* (from Equation 9) for all instances and is a real value in the range 0..1. Thus, the LCV portion of CVC can be thought of as providing the whole part of the score, with confidence providing the fractional part. Dividing *numCorrect* + *avgConf* by n + 1 results in a score in the range 0..1, as would also be obtained by averaging Equation 10 for all instances in T.

This metric gives LCV the ability to make decisions by itself unless multiple parameter settings are tied, in which case the confidence makes the decision. There will still be a bias towards giving the nearest neighbor more weight but only when LCV cannot determine which parameter setting yields better leave-one-out accuracy.

4. FIBL Learning Algorithm

This section describes the learning algorithm used by the *Fuzzy Instance-Based Learning* (FIBL) algorithm.

FIBL begins by finding the first k nearest neighbors of every instance i, where k is set to *maxk*, the maximum value of k being considered. (In our experiments we used *maxk* = 30 to leave a wide margin of error, and values of k greater than 10 were rarely if ever chosen by the system.) The nearest neighbors of each instance i, notated $i.n_1 ... i.n_{maxk}$, are stored in a list ordered from nearest to furthest for each instance, so that $i.n_1$ is the nearest neighbor of i and $i.n_k$ is the kth nearest neighbor. The distance $i.d_j$ to each of instance i's neighbors is also stored to avoid continuously recomputing this distance. This step takes $O(mn^2)$ time, where m is the number of input attributes and n is the number of instances in the training set.

CVC is used in the FIBL system to automatically find good values for the parameters k, w_k, *shape*, and *avgk*. Note that none of these parameters affect the distance between neighbors but only affect the amount of voting weight each neighbor gets. Thus, changes in these parameters can be made without requiring a new search for nearest neighbors or even an update to the stored distance to each neighbor. This allows a set of parameter values to be evaluated in $O(kn)$ time instead of the $O(mn^2)$ time required by a naive application of leave-one-out cross-validation.

To evaluate a set of parameter values, cvc_i as defined in Equation 10 is computed as follows. For each instance i, the voting weight for each of its k nearest neighbors is found according to its stored distance and the current settings of k, w_k, *shape* and *avgk*, as described in Section 2. These weights are summed in their respective classes, and the confidence of the correct class is found as in Equation 9. If the majority class is the same as the true output class of instance i, cv in Equation 10 is 1. Otherwise, it is 0. The average value of cvc_i over all n instances is used to determine the fitness of the parameter values.

The search for parameter values proceeds in a greedy manner as follows. For each iteration, one of the four parameters is chosen for adjustment, with the restriction that no parameter can be chosen twice in a row, since doing so would simply rediscover the same parameter value. The chosen parameter is set to various values as explained below while the remaining parameters are held constant. For each setting of the chosen parameter, the CVC fitness for the system is calculated, and the value that achieves the highest fitness is chosen as the new value for the parameter.

At that point, another iteration begins, in which a different parameter is chosen at random and the process is repeated until 10 attempts at tuning parameters does not improve the best CVC fitness found so far. In practice, only a few iterations are required to find good settings, after which improvements cease and the search soon terminates. The set of parameters that yield the best CVC fitness found at any point during the search are used by FIBL for classification. The four parameters are tuned as follows.

1. Choosing k. To pick a value of k, all values from 2 to $maxk$ (=30 in our experiments) are tried, and the one that results in maximum CVC fitness is chosen. Using the value $k = 1$ would make all of the other parameters irrelevant, thus preventing the system from tuning them, so only values 2 through 30 are used until all iterations are complete.

2. Choosing a shape function. Picking a vote-weighting shape function proceeds in a similar manner. The shapes *linear*, *gaussian*, and *exponential* are tried, and the shape that yields the highest CVC fitness is chosen. Using *majority* voting would make the parameters wk and $avgk$ irrelevant, so this setting is not used until all iterations are complete. At that point, majority voting is tried with values of k from 1 to 30 to test both $k = 1$ and majority voting in general, to see if either can improve upon the tuned set of parameters.

3. Setting *avgk*. Selecting a value for the flag *avgk* consists of simply trying both settings, i.e., using D_k and D_k' and seeing which yields higher CVC fitness.

4. Searching for w_k. Finding a value for w_k is more complicated because it is a real-valued parameter. The search for a good value of w_k begins by dividing the range 0..1 into ten subdivisions and trying all eleven endpoints of these divisions. For example, on the first pass, the values 0, .1, .2, ..., .9, and 1.0 are used. The value that yields the highest CVC fitness is chosen, and the range is narrowed to cover just one division on either side of the chosen value, with the constraint that the range cannot go outside of the range 0..1. For example, if .3 is chosen in the first round, then the new range is from .2 to .4. The process is repeated three times, at which point the effect on classification becomes negligible.

Pseudo-code for the parameter-finding portion of the learning algorithm is shown in Figure 2. This routine assumes that the nearest *maxk* neighbors of each instance T have been found and returns the parameters that yield the highest CVC fitness found during the search. Once these parameters have been found, the neighbor lists can be discarded, and only the raw instances and best parameters need to be retained for use during subsequent classification.

```
FindParams(maxTime, training set T): bestParams
    Assume that the maxk nearest neighbors have been
        found for each instance i in T.
    Let timeSinceImprovement=0.
    Let bestCVC=0.
    While timeSinceImprovement < maxTime
        Choose a random parameter p to adjust.
        If (p="k") try k=2..30, and set k to best value found.
        If (p="shape") try linear, gaussian, and exponential.
        If (p="avgk") try D_k and D'_k.
        If (p="w_k")
            Let min=0 and max=1
            For iteration=1 to 3
                Let width=(min-max)/10.
                Try w_k=min..max in steps of width.
                Let min=best w_k-width (if min<0, let min=0)
                Let max=best w_k+width (if max>1, let max=1)
            Endfor
        If bestCVC was improved during this iteration,
            then let timeSinceImprovement=0,
            and let bestParams=current parameter settings.
    Endwhile.
    Let shape=majority, and try k=1..30.
    if bestCVC was improved during this search,
        then let bestParams=current parameter settings.
    Return bestParams.
```

Figure 2. Learning algorithm for FIBL.

In Figure 2, to "try" a parameter value means to set the parameter to that value, find the CVC fitness of the system, and, if the fitness is better than any seen so far, set *bestCVC* to this fitness, and remember the current set of parameter values in *bestParams*.

Trying a parameter value takes kn time, and tuning k, *shape*, *avgk*, and w_k test 29, 3, 2, and 30 values, respectively. The entire process takes O(knt) time, where t is the number of iterations required before the stopping criterion is met. In practice t is small (e.g., less than 20), since tuning each parameter once or twice is usually sufficient. These time requirements are quite acceptable, especially compared to algorithms that require repeated O(mn^2) steps.

The time spent tuning parameters is done just once during learning, and is dominated by the O(mn^2) step required to find the nearest neighbors of each instance. During execution, classification takes O(mn) time, which is the same as the basic nearest neighbor rule.

5. Experimental Results

The Fuzzy Instance-Based Learning (FIBL) algorithm was implemented and tested on 31 applications from the Machine Learning Database Repository at the University of California, Irvine [Merz & Murphy, 1996]. FIBL was compared to a static instance-based learning algorithm that is identical to FIBL except that it uses $k = 3$ and majority voting and thus does not fine-tune parameters. FIBL was also compared to an otherwise identical algorithm that uses leave-one-out cross-validation (LCV) instead of CVC to decide on the various parameters.

For each dataset each algorithm was trained using 90% of the available data. The remaining 10% of the data was classified using the instances in T and the best parameter settings found during training. The average accuracy over 10 such trials (i.e., 10-fold cross-validation accuracy) is reported for each dataset in Table 1.

Dataset	Static	LCV	FIBL
Anneal	93.11	94.49	**94.62**
Australian	84.78	85.08	**85.36**
Breast Cancer(WI)	96.28	**96.71**	96.42
Bridges	**66.09**	65.09	65.09
Crx	83.62	84.78	**85.07**
Echocardiogram	94.82	**96.07**	**96.07**
Flag	61.34	62.39	**63.37**
Glass	**73.83**	67.81	69.20
Heart	81.48	81.85	**83.34**
Heart(Cleveland)	81.19	81.48	**83.15**
Heart(Hungarian)	79.55	80.60	**80.93**
Heart(Long Beach)	70.00	**73.50**	73.00
Heart(More)	73.78	77.03	**78.52**
Heart(Swiss)	**92.69**	92.63	92.63
Hepatitis	80.62	**83.00**	81.79
Horse Colic	57.84	59.51	**65.15**
Image Segmentation	**93.10**	91.67	91.91
Ionosphere	84.62	**86.91**	86.62
Iris	94.00	**95.33**	**95.33**
LED Creator+17	67.10	71.80	**71.90**
LED Creator	**73.40**	72.30	72.90
Liver (Bupa)	**65.57**	61.77	61.41
Pima Diabetes	73.56	73.58	**75.26**
Promoters	93.45	**94.09**	93.09
Sonar	**87.55**	83.57	84.10
Soybean (Large)	88.59	90.54	**90.86**
Vehicle	71.76	**72.13**	71.54
Voting	95.64	**95.85**	**95.85**
Vowel	96.57	**98.29**	**98.29**
Wine	94.93	96.01	**97.71**
Zoo	94.44	94.44	**97.78**
Average	**82.11**	**82.59**	**83.17**

Table 1. Generalization accuracy of IBL algorithms using static majority voting (*Static*), cross-validation to make decisions (*LCV*), and CVC as used in FIBL.

FIBL (using CVC) had the highest generalization accuracy in 18 out of these 31 datasets, LCV was highest in 10 datasets and the static majority-voting algorithm was highest in 7 cases. FIBL was an average of over 1% higher than the static algorithm in generalization accuracy on these datasets. LCV fell almost exactly halfway between the static and FIBL methods, indicating that CVC improves upon the performance of LCV, which in turn improves upon the performance of default settings. All of these algorithms have substantially higher generalization accuracy than the basic nearest neighbor rule using a Euclidean distance function [Wilson & Martinez, 1997].

6. Related Work

The standard k-nearest neighbor algorithm uses majority voting. Dudani [1976] proposed a distance-weighted nearest neighbor algorithm that uses linear voting similar to Equation (2) with $w_k = 0$, except that the denominator is D_k - D_1, so that the first nearest neighbor gets a weight of 1 regardless of its distance. Since the kth neighbor gets 0 weight, Dudani's model really uses at most k-1 neighbors to vote on the output class.

Keller, Gray & Givens [1985] proposed a *Fuzzy K-Nearest Neighbor Algorithm* that also uses a distance-weighted voting scheme. Their distance-weighting function is

$$w_i = D_i^{-2/(m-1)} \tag{11}$$

where m is a user-supplied parameter (they used $m = 2$ in their experiments). This is similar in shape to the exponentially weighted function given in Equation (4). However, it differs in that the rate of drop-off is not influenced by the data density, and the weight approaches infinity as the distance approaches 0.

In the Fuzzy k-NN system the *class membership* in each class is the voting weight for that class divided by the total voting weight for all classes, similar to the confidence levels used by FIBL. They point out that the class membership can be used to indicate how confident the system is in its output. However, if applied to the selection of k and other parameters, the Fuzzy k-NN system would have the same problem of favoring any parameters that increase the weight of nearer neighbors, regardless of their effect on classification accuracy, as discussed in Section 3.2.

The *IB4* algorithm [Aha, 1992] is an instance-based learning algorithm that was designed to handle irrelevant attributes. It also finds class memberships similar to FIBL. IB4 maintains a separate set of attribute weights for each output class, and also maintains a separate subset S_c of the training set T for each output class c. Instances are considered either "members" or "non-members" of each class, and IB4 computes class membership as

$$Membership(x,c) = \frac{Similarity(c,x,neg)}{Similarity(c,x,neg) + Similarity(c,x,pos)} \tag{12}$$

where x is the input being classified, c is the class, *neg* is the nearest "acceptable" instance of a class other than c, and *pos* is the nearest "acceptable" instance of class c. An instance is *acceptable* if it passes a statistical test based on how accurate it has classified the instances seen so far.

7. Conclusions and Future Research Directions

The *Fuzzy Instance-Based Learning* (FIBL) system combines the use of cross-validation accuracy and confidence to generate an evaluation function that returns real-valued differences in fitness in response to even small changes in parameters. It avoids the problem of frequent ties that occurs when using cross-validation alone. It also does not suffer from the strong bias towards heavily weighting nearer neighbors that occurs when using confidence alone.

In our experiments on a collection of 31 datasets, FIBL was able to successfully use the new CVC evaluation method in conjunction with a distance-weighted voting scheme to improve average accuracy over a static majority-voting algorithm or a distance-weighted algorithm using only cross-validation to make decisions.

Future research will combine FIBL with instance pruning models to reduce storage and improve classification speed.

References

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.

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

Batchelor, Bruce G., (1978). *Pattern Recognition: Ideas in Practice*. New York: Plenum Press, pp. 71-72.

Biberman, Yoram, (1994). A Context Similarity Measure. *In Proceedings of the European Conference on Machine Learning (ECML-94)*. Catalina, Italy: Springer Verlag, pp. 49-63.

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

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.

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

Diday, Edwin, (1974). Recent Progress in Distance and Similarity Measures in Pattern Recognition. *Second International Joint Conference on Pattern Recognition*, pp. 534-539.

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).*

Dudani, Sahibsingh A., (1976). "The Distance-Weighted *k*-Nearest-Neighbor Rule," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, no. 4, April 1976, pp. 325-327.

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.

Keller, James M., Michael R. Gray, and James A. Givens, Jr., (1985). "A Fuzzy *K*-Nearest Neighbor Algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, no. 4, July/August 1985, pp. 580-585.

Kohavi, Ron, (1995). "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95).*

Lebowitz, Michael, (1985). "Categorizing Numeric Information for Generalization," *Cognitive Science*, vol. 9, pp. 285-308.

Merz, C. J., and P. M. Murphy, (1996). *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: http://www.ics.uci.edu/~mlearn/ MLRepository.html.

Michalski, Ryszard S., Robert E. Stepp, and Edwin Diday, (1981). A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts. *Progress in Pattern Recognition*, vol. 1, Laveen N. Kanal and Azriel Rosenfeld (Eds.). New York: North-Holland, pp. 33-56.

Moore, Andrew W., and Mary S. Lee, (1993). "Efficient Algorithms for Minimizing Cross Validation Error," In *Machine Learning: Proceedings of the Eleventh International Conference*, Morgan Kaufmann.

Nadler, Morton, and Eric P. Smith, (1993). *Pattern Recognition Engineering*. New York: Wiley, pp. 293-294.

Rachlin, John, Simon Kasif, Steven Salzberg, David W. Aha, (1994). "Towards a Better Understanding of Memory-Based and Bayesian Classifiers," in *Proceedings of the Eleventh International Machine Learning Conference,* New Brunswick, NJ: Morgan Kaufmann, pp. 242-250.

Salzberg, Steven, (1991). "A Nearest Hyperrectangle Learning Method," *Machine Learning*, vol. 6, pp. 277-309.

Schaffer, Cullen, (1993). "Selecting a Classification Method by Cross-Validation," *Machine Learning*, vol. 13, no. 1.

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

Tversky, Amos, (1977). Features of Similarity. *Psychological Review*, vol. 84, no. 4, pp. 327-352.

Ventura, Dan, and Tony R. Martinez (1995). "An Empirical Comparison of Discretization Methods." *In Proceedings of the Tenth International Symposium on Computer and Information Sciences*, pp. 443-450.

Watson, I., and F. Marir, (1994). "Case-Based Reasoning: A Review," *The Knowledge Engineering Review*, vol. 9, no. 4.

Wettschereck, Dietrich, and Thomas G. Dietterich, (1995). "An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms," *Machine Learning*, vol. 19, no. 1, pp. 5-28.

Wilson, D. Randall, and Tony R. Martinez, (1997). "Improved Heterogeneous Distance Functions," *Journal of Artificial Intelligence Research*, vol. 6, no. 1, pp. 1-34.

Zadeh, Lotfi A., (1965). "Fuzzy Sets," *Information and Control*, vol. 8, pp. 338-353.

Part V

Conclusion

"Better is the end of a thing than the beginning thereof."
—Ecclesiastes 7:8

As discussed in the introduction in Chapter 1, the basic nearest neighbor algorithm has had success in some domains but suffers from inadequate distance functions, large storage requirements, slow execution speed, a sensitivity to noise, and an inability to adjust decision boundaries after storing the data.

This dissertation includes several enhancements to the nearest neighbor algorithm that address several of these problems individually. However, each enhancement has been tested largely in absence of the other enhancements so that the effect of each such modification could be more closely analyzed.

Chapter 10 presents the *Integrated Decremental Instance-Based Learning* (IDIBL) algorithm, which combines the most successful elements of each of the preceding chapters into a comprehensive learning system. This system is compared to earlier, less enhanced versions of the algorithm to show that the combination of improvements yields better generalization accuracy than the individual improvements alone. Finally, IDIBL is compared to 16 other major machine learning and neural network algorithms.

Since Chapter 10 combines enhancements from earlier chapters in the dissertation, several sections are somewhat redundant and are optional reading for those familiar with earlier chapters. Specifically, Section 2 presents the IVDM distance metric from Chapter 5; Section 3 presents the DROP4 algorithm from Chapter 7; and Section 4 presents the distance-weighted voting and cross-validation/confidence level accuracy estimation metric from Chapter 9. This redundancy is necessary to allow Chapter 10 to be published as a stand-alone document. However, it does show in Section 5 how these individual techniques can be combined into a comprehensive system, and in Section 6 provides new empirical results.

Chapter 10 can be referenced as

Wilson, D. Randall, and Tony R. Martinez, (1997). "Advances in Instance-Based Learning," submitted to *Machine Learning Journal*.

Chapter 11 provides overall conclusions to the dissertation and gives potential areas for future research.

Chapter 10

Advances in Instance-Based Learning

"...then will I cause the good and bad to be gathered;
and the good will I preserve unto myself,
and the bad will I cast away into its own place.
And then cometh the season and the end...."
—Jacob 5:77

Submitted to *Machine Learning Journal*, 1997.

Abstract

The basic nearest-neighbor rule generalizes well in many domains but has several shortcomings, including inappropriate distance functions, large storage requirements, slow execution time, sensitivity to noise, and an inability to adjust its decision boundaries after storing the training data. This paper proposes methods for overcoming each of these weaknesses and combines these methods into a comprehensive learning system called the *Integrated Decremental Instance-Based Learning Algorithm* (IDIBL) that seeks to reduce storage, improve execution speed, and increase generalization accuracy, when compared to the basic nearest neighbor algorithm and other learning models. In our experiments IDIBL achieves higher generalization accuracy than other less comprehensive instance-based learning algorithms, while requiring less than one-fifth the storage of the nearest neighbor algorithm and improving execution speed by a corresponding factor. In experiments on 31 datasets, IDIBL also achieves higher generalization accuracy than those reported for 16 major machine learning and neural network models.

1. Introduction

The *Nearest Neighbor* algorithm [Cover & Hart, 1967; Dasarathy, 1991] is an inductive learning algorithm that stores all of the n available training examples (*instances*) from a *training set, T,* during learning. Each instance has an *input vector* $\boldsymbol{x}$, and an *output class c*. During *generalization*, these systems use a distance function to determine how close a new input vector $\boldsymbol{y}$ is to each stored instance, and use the nearest instance or instances to predict the output class of $\boldsymbol{y}$ (i.e., to *classify* $\boldsymbol{y}$).

The nearest neighbor algorithm is intuitive and easy to understand, it learns quickly, and it provides good generalization accuracy for a variety of real-world classification tasks (*applications*).

However, in its basic form, the nearest neighbor algorithm has several weaknesses:

- Its distance functions are often inappropriate or inadequate for applications with both linear and nominal attributes.

- It has large storage requirements, because it stores all of the available training data in the model.
- It is slow during execution, because all of the training instances must be searched in order to classify each new input vector.
- Its accuracy degrades rapidly with the introduction of noise.
- Its accuracy degrades with the introduction of irrelevant attributes.
- It has no ability to adjust its decision boundaries after storing the training data.

Many researchers have developed extensions to the nearest neighbor algorithm, which are commonly called *instance-based* learning algorithms [Aha, Kibler & Albert, 1991; Aha, 1992; Dasarathy, 1991]. Related paradigms include *memory-based reasoning* methods [Stanfill & Waltz, 1986; Cost & Salzberg, 1993; Rachlin et al., 1994], *exemplar-based generalization* [Salzberg, 1991; Wettschereck & Dietterich, 1995], and *case-based reasoning* (CBR) [Watson & Marir, 1994].

Some efforts in instance-based learning and related areas have focused on one or more of the above problems without addressing them all in a comprehensive system. Others have used solutions to some of the problems that were not as robust as those used by others.

The authors have proposed several extensions to instance-based learning algorithms as well [Wilson & Martinez, 1996, 1997a,b,c], and have purposely focused on only one or only a few of the problems at a time so that the effect of each proposed extension could be evaluated independently, based on its own merits.

This paper proposes a comprehensive learning system, called the *Integrated Decremental Instance-Based Learning* (IDIBL) algorithm, that combines successful extensions from earlier work to overcome the weaknesses of the basic nearest neighbor rule mentioned above.

Section 2 discusses the need for a robust heterogeneous distance function and describes the *Interpolated Value Distance Metric*. Section 3 presents a *decremental* pruning algorithm (i.e., one that starts with the entire training set and removes instances that are not needed) that reduces the number of instances stored in the system and thus decreases storage requirements while increasing classification speed. It also reduces the sensitivity of the system to noise. Section 4 presents a distance-weighting scheme and shows how cross-validation and confidence can be used to provide a more flexible concept description and to allow the system to be fine-tuned. Section 5 presents the learning algorithm for IDIBL, and shows how it operates during classification.

Section 6 presents empirical results that indicate how well IDIBL works in practice. IDIBL is first compared to the basic nearest neighbor algorithm and several extensions to it. It is then compared to results reported for 16 major machine learning and neural network models on 31 datasets. IDIBL achieves the highest average generalization accuracy of any of the models examined in these experiments.

2. Heterogeneous Distance Functions

There are many learning systems that depend upon a good distance function to be successful, including the instance-based learning algorithms and the related models mentioned in the introduction. In addition, many neural network models also make use of distance functions, including radial basis function networks [Broomhead & Lowe, 1988; Renals & Rohwer, 1989; Wasserman, 1993], counterpropagation networks [Hecht-Nielsen, 1987], ART [Carpenter & Grossberg, 1987], self-organizing maps [Kohonen, 1990] and competitive learning [Rumelhart & McClelland, 1986]. Distance functions are also used in many fields besides machine learning and neural networks, including statistics [Atkeson, Moore & Schaal, 1996], pattern recognition [Diday, 1974; Michalski, Stepp & Diday, 1981], and cognitive psychology [Tversky, 1977; Nosofsky, 1986].

2.1. Linear Distance Functions

A variety of distance functions are available for such uses, including the Minkowsky [Batchelor, 1978], Mahalanobis [Nadler & Smith, 1993], Camberra, Chebychev, Quadratic, Correlation, and Chi-square distance metrics [Michalski, Stepp & Diday, 1981; Diday, 1974]; the Context-Similarity measure [Biberman, 1994]; the Contrast Model [Tversky, 1977]; hyperrectangle distance functions [Salzberg, 1991; Domingos, 1995] and others.

Although there have been many distance functions proposed, by far the most commonly used is the Euclidean Distance function, which is defined as

$$E(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{a=1}^{m} (x_a - y_a)^2} \qquad (1)$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are two input vectors (one typically being from a stored instance, and the other an input vector to be classified) and m is the number of input variables (*attributes*) in the application.

None of the above distance functions is designed to handle applications with both linear and nominal attributes. A *nominal* attribute is a discrete attribute whose values are not necessarily in any linear order. For example, a variable representing color might have values such as *red, green, blue, brown, black* and *white*, which could be represented by the integers 1 through 6, respectively. Using a linear distance measurement such as Euclidean distance on such values makes little sense in this case.

Some researchers have used the *overlap* metric for nominal attributes and normalized Euclidean distance for linear attributes [Aha, Kibler & Albert, 1991; Aha, 1992; Giraud-Carrier & Martinez, 1995]. The overlap metric uses a distance of 1 between attribute values that are different, and a distance of 0 if the values are the same. This metric loses much of the information that can be found from the nominal attribute values themselves.

2.2. Value Difference Metric for Nominal Attributes

The Value Difference Metric (VDM) was introduced by Stanfill and Waltz [1986] to provide an appropriate distance function for nominal attributes. A simplified version of the VDM (without the weighting schemes) defines the distance between two values x and y of an attribute a as

$$vdm_a(x,y) = \sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^q = \sum_{c=1}^{C} \left| P_{a,x,c} - P_{a,y,c} \right|^q \tag{2}$$

where

- $N_{a,x}$ is the number of instances in the training set T that have value x for attribute a;
- $N_{a,x,c}$ is the number of instances in T that have value x for attribute a and output class c;
- C is the number of output classes in the problem domain;
- q is a constant, usually 1 or 2; and
- $P_{a,x,c}$ is the conditional probability that the output class is c given that attribute a has the value x, i.e., $P(c \mid x_a)$. As can be seen from (2), $P_{a,x,c}$ is defined as

$$P_{a,x,c} = \frac{N_{a,x,c}}{N_{a,x}} \tag{3}$$

where $N_{a,x}$ is the sum of $N_{a,x,c}$ over all classes, i.e.,

$$N_{a,x} = \sum_{c=1}^{C} N_{a,x,c} \tag{4}$$

and the sum of $P_{a,x,c}$ over all C classes is 1 for a fixed value of a and x.

Using the distance measure $vdm_a(x,y)$, two values are considered to be closer if they have more similar classifications (i.e., more similar correlations with the output classes), regardless of what order the values may be given in. In fact, linear discrete attributes can have their values remapped randomly without changing the resultant distance measurements.

One problem with the formulas presented above is that they do not define what should be done when a value appears in a new input vector that never appeared in the training set. If attribute a never has value x in any instance in the training set, then $N_{a,x,c}$ for all c will be 0, and $N_{a,x}$ (which is the sum of $N_{a,x,c}$ over all classes) will also be 0. In such cases $P_{a,x,c} = 0/0$, which is undefined. For nominal attributes, there is no way to know what the probability should be for such a value, since there is no inherent ordering to the values. In this paper we assign $P_{a,x,c}$ the default value of 0 in such cases (though it is also possible to let $P_{a,x,c} = 1/C$, where C is the number of output classes, since the sum of $P_{a,x,c}$ for $c = 1..C$ is always 1.0).

If this distance function is used directly on continuous attributes, the values can all potentially be unique, in which case $N_{a,x}$ is 1 for every value x, and $N_{a,x,c}$ is 1 for one value of c and 0 for all others for a given value x. In addition, new vectors are likely to have unique values, resulting in the division by zero problem above. Even if the value of 0 is substituted for 0/0, the resulting distance measurement is nearly useless.

Even if all values are not unique, there are often enough different values for a continuous attribute that the statistical sample is unreliably small for each value, and the distance measure is still untrustworthy. Because of these problems, it is inappropriate to use the VDM directly on continuous attributes.

Previous systems such as PEBLS [Cost & Salzberg, 1993; Rachlin et al., 1994] that have used VDM or modifications of it have typically relied on *discretization* [Lebowitz, 1985], which can degrade accuracy [Wilson & Martinez, 1997a].

2.3. Interpolated Value Difference Metric

Since the Euclidean distance function is inappropriate for nominal attributes, and the VDM function is inappropriate for direct use on continuous attributes, neither is appropriate for applications with both linear and nominal attributes. This section presents the *Interpolated Value Difference Metric* (IVDM) [Wilson & Martinez, 1997a] that allows VDM to be applied directly to continuous attributes.

The original value difference metric (VDM) uses statistics derived from the training set instances to determine a probability $P_{a,x,c}$ that the output class is c given the input value x for attribute a.

When using IVDM, continuous values are discretized into s equal-width intervals (though the continuous values are also retained for later use), where s is chosen to be 5 or C, whichever is greatest, where C is the number of output classes in the problem domain.

The width w_a of a discretized interval for attribute a is given by

$$w_a = \frac{|max_a - min_a|}{s} \tag{5}$$

where max_a and min_a are the maximum and minimum value, respectively, occurring in the training set for attribute a. The discretized value v of a continuous value x for attribute a is an integer from 1 to s, and is given by

$$v = discretize_a(x) = \begin{cases} x, \text{ if } a \text{ is discrete, else} \\ s, \text{ if } x = \max_a, \text{ else} \\ \lfloor (x - \min_a) / w_a \rfloor + 1 \end{cases} \tag{6}$$

After deciding upon s and finding w_a, the discretized values of continuous attributes can be used just like discrete values of nominal attributes in finding $P_{a,x,c}$. Figure 1 lists pseudo-code for how this is done.

The distance function for the Interpolated Value Difference Metric is defined as

$$IVDM(\boldsymbol{x}, \boldsymbol{y}) = \sum_{a=1}^{m} ivdm_a(x_a, y_a)^2 \tag{7}$$

where $ivdm_a$ is defined as

$$ivdm_a(x,y) = \begin{cases} vdm_a(x,y) & \text{if } a \text{ is discrete} \\ \sum_{c=1}^{C} \left| p_{a,c}(x) - p_{a,c}(y) \right|^2 & \text{otherwise} \end{cases} \tag{8}$$

```
FindProbabilities(training set T)
    For each attribute a
        For each instance i in T
            Let x be the input value for attribute a of instance i.
            v = discretize_a(x) [which is just x if a is discrete]
            Let c be the output class of instance i.
            Increment N_a,v,c by 1.
            Increment N_a,v by 1.
        For each discrete value v (of attribute a)
            For each class c
                If N_a,v=0
                    Then P_a,v,c=0
                    Else P_a,v,c = N_a,v,c / N_a,v
    Return 3-D array P_a,v,c.
```

Figure 1. Pseudo code for finding $P_{a,x,c}$.

Unknown input values [Quinlan, 1989] are treated as simply another discrete value, as was done in [Domingos, 1995]. The formula for determining the interpolated probability value $p_{a,c}(x)$ of a continuous value x for attribute a and class c is

$$p_{a,c}(x) = P_{a,u,c} + \left(\frac{x - mid_{a,u}}{mid_{a,u+1} - mid_{a,u}} \right) * (P_{a,u+1,c} - P_{a,u,c}) \tag{9}$$

In this equation, $mid_{a,u}$ and $mid_{a,u+1}$ are midpoints of two consecutive discretized ranges such that $mid_{a,u} \leq x < mid_{a,u+1}$. $P_{a,u,c}$ is the probability value of the discretized range u, which is taken to be the probability value of the midpoint of range u (and similarly for $P_{a,u+1,c}$). The value of u is found by first setting $u = discretize_a(x)$, and then subtracting 1 from u if $x < mid_{a,u}$. The value of $mid_{a,u}$ can then be found as follows.

$$mid_{a,u} = min_a + width_a * (u+.5) \tag{10}$$

Figure 2 shows the values of $p_{a,c}(x)$ for attribute a=1 of the *Iris* database for all three output classes (i.e. c=1, 2, and 3). Since there are no data points outside the range $min_a..max_a$, the probability value $P_{a,u,c}$ is taken to be 0 when $u < 1$ or $u > s$, which can be seen visually by the diagonal lines sloping toward zero on the outer edges of the graph. Note that the sum of the probabilities for the three output classes sum to 1.0 at every point from the midpoint of range 1 through the midpoint of range 5.

In experiments reported by the authors [Wilson & Martinez, 1997a] on 48 datasets from the UCI machine learning databases [Merz & Murphy, 1996], a nearest neighbor classifier using IVDM was able to achieve almost 5% higher generalization accuracy on average over a nearest neighbor classifier using either Euclidean distance or the Euclidean-Overlap metric mentioned in Section 2.1. It also achieved higher accuracy than a classifier that used discretization on continuous attributes in order to use the VDM distance function.

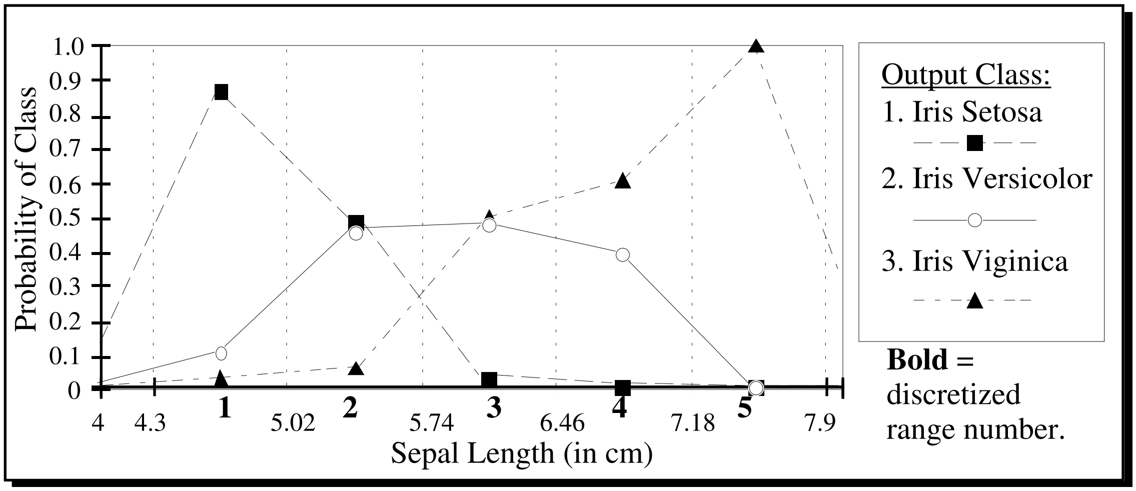


Figure 2. Interpolated probability values for attribute 1 of the *Iris* database.

3. Instance Pruning Techniques

One of the main disadvantages of the basic nearest neighbor rule is that it has large storage requirements because it stores all n instances in the training set T in memory. It also has slow execution speed because it must find the distance between a new input vector and each of the n instances in order to find the nearest neighbor(s) of the new input vector, which is necessary for classification. In addition, since it stores every instance in the training set, noisy instances (i.e., those with errors in the input vector or output class, or those not representative of typical cases) are stored as well, which can degrade generalization accuracy.

3.1. Speeding Classification

It is possible to use k-dimensional trees ("k-d trees") [Wess, Althoff & Derwand, 1994; Sproull, 1991; Deng & Moore, 1995] to find the nearest neighbor in O(log$n$) time in the best case. However, as the dimensionality grows, the search time can degrade to that of the basic nearest neighbor rule.

Another technique used to speed the search is *projection* [Papadimitriou & Bentley, 1980], where instances are sorted once by each dimension, and new instances are classified by searching outward along each dimension until it can be sure the nearest neighbor has been found. Again, an increase in dimensionality reduces the effectiveness of the search.

Even when these techniques are successful in reducing execution time, they do not reduce storage requirements. Also, they do nothing to reduce the sensitivity of the classifier to noise.

3.2. Reduction Techniques

One of the most straightforward ways to speed classification in a nearest-neighbor system is by removing some of the instances from the instance set. This also addresses another of the main disadvantages of nearest-neighbor classifiers—their large storage requirements. In addition, it is sometimes possible to remove noisy instances from the instance set and actually improve generalization accuracy.

A large number of such reduction techniques have been proposed, including the *Condensed Nearest Neighbor Rule* [Hart, 1968], *Selective Nearest Neighbor Rule* [Ritter et. al., 1975], the *Reduced Nearest Neighbor Rule* [Gates, 1972], the *Edited Nearest Neighbor* [Wilson, 1972], the *All k-NN* method [Tomek, 1976], *IB2* and *IB3* [Aha, Kibler & Albert, 1991; Aha, 1992], the *Typical Instance Based Learning* Zhang [1992], *random mutation hill climbing* [Skalak, 1994; Papadimitriou & Steiglitz, 1982], and instance selection by *encoding length heuristic* [Cameron-Jones, 1995]. Other techniques exist that modify the original instances and use some other representation to store exemplars, such as prototypes [Chang, 1974]; rules, as in RISE 2.0 [Domingos, 1995]; hyperrectangles, as in EACH [Salzberg, 1991]; and hybrid models [Dasarathy, 1979; Wettschereck, 1994].

These and other reduction techniques are surveyed in depth in [Wilson & Martinez, 1997c], along with several new reduction techniques called *DROP1-DROP5*.

3.3. DROP4 Reduction Algorithm

This section presents an algorithm called the *Decremental Reduction Optimization Procedure 4* (DROP4) [Wilson & Martinez, 1997c] that is used by IDIBL to reduce the number of instances that must be stored in the final system and to correspondingly decrease classification time. DROP4 also makes IDIBL more robust in the presence of noise. This procedure is *decremental*, meaning that it begins with the entire training set, and then removes instances that are deemed unneccesary. This is different than the *incremental* approaches that begin with an empty subset S and add instances to it, as is done by IB3 [Aha, Kibler & Albert, 1991] and several other instance-based algorithms.

DROP4 uses the following basic rule to decide if it is safe to remove an instance i from the instance set S (where $S = T$ originally).

Remove instance i from S if at least as many of its associates in T
would be classified correctly without i.

To see if an instance i can be removed using this rule, each *associate* (i.e., each instance that has i as one of its neighbors) is checked to see what effect the removal of i would have on it.

Removing i causes each associate $i.a_j$ to use its k+1st nearest neighbor ($i.a_j.n_{k+1}$) in S in place of i. If i has the same class as $i.a_j$, and $i.a_j.n_{k+1}$ has a different class than $i.a_j$, this weakens its classification and could cause $i.a_j$ to be misclassified by its neighbors. On the other hand, if i is a different class than $i.a_j$ and $i.a_j.n_{k+1}$ is the same class as $i.a_j$, the removal of i could cause a previously misclassified instance to be classified correctly.

In essence, this rule tests to see if removing i would degrade leave-one-out cross-validation accuracy, which is an estimate of the true generalization ability of the resulting classifier. An instance is removed when it results in the same level of generalization with lower storage requirements. By maintaining lists of k+1 neighbors and an average of k+1 associates (and their distances), the leave-one-out cross-validation can be computed in O(k) time for each instance instead of the usual O(mn) time, where n is the number of instances in the training set and m is the number of input attributes. An O(mn) step is only required once an instance is selected for removal.

The algorithm for DROP4 proceeds as shown in Figure 3.

```
DROP4(Training set T): Instance set S.
    Let S = T.
    For each instance i in S:
        (Assume we know i.n1...i.nk+1, the k+1 nearest neighbors of i in S.)
        Add i to each of its neighbors' lists of associates.
    For each instance i in S:
        Let with = # of associates of i classified correctly with i as a neighbor.
        Let without = # of associates of i classified correctly without i.
        If (without - with) ≥ 0
            Remove i from S.
            For each associate a of i
                Remove i from a's list of nearest neighbors
                Find a new nearest neighbor for a.
                Add a to its new neighbor's list of associates.
        Endif
    Return S.
```

Figure 3: Pseudo-code for DROP4.

This algorithm assumes that a list of nearest neighbors for each instance has already been found (as explained in Section 5), and begins by making sure each neighbor has a list of its associates. Then each instance in S is removed if its removal does not hurt the classification of the instances remaining in S. When an instance i is removed, all of its associates must remove i from their list of nearest neighbors and then must find a new nearest neighbor $a.n_j$ so that they still have k+1 neighbors in their list. When they find a new neighbor $a.n_j$, they also add themselves to $a.n_j$'s list of associates so that at all times every instance has a current list of neighbors and associates.

Each instance i in the original training set T continues to maintain a list of its $k + 1$ nearest neighbors in S, even after i is removed from S. This in turn means that instances in S have associates that are both in and out of S, while instances that have been removed from S have no associates (because they are no longer a neighbor of any instance).

This algorithm removes noisy instances, because a noisy instance i usually has associates that are mostly of a different class, and such associates will be at least as likely to be classified correctly without i. DROP4 also removes an instance i in the center of a cluster because associates there are not near instances of other classes, and thus continue to be classified correctly without i.

Near the border, the removal of some instances can cause others to be classified incorrectly because the majority of their neighbors can become enemies. Thus this algorithm tends to keep non-noisy border points. At the limit, there is typically a collection of border instances such that the majority of the k nearest neighbors of each of these instances is the correct class.

The order of removal can be important to the success of a reduction algorithm. DROP4 initially sorts the instances in S by the distance to their nearest *enemy*, which is the nearest neighbor of a different class. Instances are then checked for removal beginning at the instance furthest from its nearest enemy. This tends to remove instances furthest from the decision boundary first, which in turn increases the chance of retaining border points.

However, noisy instances are also "border" points, so they can cause the order of removal to be drastically changed. In addition, it is often desirable to remove the noisy instances before any of the others so that the rest of the algorithm is not influenced as heavily by the noise.

DROP4 therefore uses a noise-filtering pass *before* sorting the instances in *S*. This is done using a rule similar to the *Edited Nearest Neighbor* rule [Wilson, 1972], which states that any instance misclassified by its *k* nearest neighbors is removed. In DROP4, however, the noise-filtering pass removes each instance only if it is (1) misclassified by its *k* nearest neighbors, *and* (2) it does not hurt the classification of its associates. This noise-filtering pass removes noisy instances, as well as close border points, which can in turn smooth the decision boundary slightly. This helps to avoid "overfitting" the data, i.e., using a decision surface that goes beyond modeling the underlying function and starts to model the data sampling distribution as well.

After removing noisy instances from *S* in this manner, the instances are sorted by distance to their nearest enemy remaining in *S*, and thus points far from the real decision boundary are removed first. This allows points internal to clusters to be removed early in the process, even if there were noisy points nearby.

In experiments reported by the authors [Wilson & Martinez, 1997c] on 31 datasets from the UCI machine learning database repository, DROP4 was compared to a *k*NN classifier that used 100% of the training instances for classification. DROP4 was able to achieve an average generalization accuracy that was just 1% below the *k*NN classifier while retaining only 16% of the original instances. Furthermore, when 10% noise was added to the output class in the training set, DROP4 was able achieve higher accuracy than *k*NN while using even less storage than before. DROP4 also compared favorably with the algorithms mentioned in Section 3.2 [Wilson & Martinez, 1997c].

4. Distance-Weighting and Confidence Levels

One disadvantage of the basic nearest neighbor classifier is that it cannot make adjustments to its decision surface after storing the data. This allows it to learn quickly, but prevents it from generalizing accurately in some cases.

Several researchers have proposed extensions that add more flexibility to instance-based systems. One of the first extensions [Cover & Hart, 1967] was the introduction of the parameter *k*, the number of neighbors that vote on the output of an input vector. A variety of other extensions have also been proposed, including various attribute-weighting schemes [Wettschereck, Aha, and Mohri, 1995; Aha, 1992; Aha & Goldstone, 1992; Mohri & Tanaka, 1994; Lowe, 1995; Wilson & Martinez, 1996], exemplar weights [Cost & Salzberg, 1993; Rachlin et al., 1994; Salzberg, 1991; Wettschereck & Dietterich, 1995], and distance-weighed voting [Dudani, 1976; Keller, Gray & Givens, 1985].

The value of *k* and other parameters are often found using *cross-validation* [Schaffer, 1993; Moore & Lee, 1993; Kohavi, 1995]. In *leave-one-out* cross-validation (LCV), each instance *i* is classified by the instances in the training set *T* other than *i* itself, so that almost all of the data is available for each classification attempt.

One problem with using CV or LCV to fine-tune a learning system (*e.g.*, when deciding whether to use $k = 1$ or $k = 3$, or when deciding what weight to give to an input attribute) is that it can yield only a fixed number of discrete accuracy estimates. Given *n* instances in a training set, CV will yield an accuracy estimation of 0 or 1 for each instance, yielding an average estimate that is in the range 0..1, but only in increments of $1/n$. This is equivalent in its usefulness to receiving an integer r in the range 0..n indicating *how many* of the instances are classified correctly using the current parameter settings.

Usually a change in any given parameter will not change r by more than a small value, so the change in r given a change in parameter, is usually a small integer such as -3...3. Unfortunately, changes in parameters often have *no* effect on r, in which case CV does not provide helpful information as to which alternative to choose. This problem occurs quite frequently in some systems and limits the extent to which CV can be used to fine-tune an instance-based learning model.

The authors proposed the *Fuzzy Instance-Based Learning* (FIBL) algorithm [Wilson & Martinez, 1997d] that uses a combination of LCV and confidence to tune parameters in a distance-weighted voting scheme. IDIBL adopts these extensions, which are explained in this section.

4.1. Distance-Weighted Voting

Let $\boldsymbol{y}$ be the input vector to be classified and let $n_1...n_k$ be the k nearest neighbors of $\boldsymbol{y}$ in a subset S (found via the pruning technique DROP4) of the training set T. Let D_j be the distance from the jth neighbor using the IVDM distance function.

In the IDIBL algorithm, the voting weight of each of the k nearest neighbors depends on its distance from the input vector $\boldsymbol{y}$. The weight is 1 when the distance is 0 and decreases as the distance grows larger. The way in which the weight decreases as the distance grows depends on which *shape function* is used. The shapes used in FIBL are: *majority, linear, gaussian,* and *exponential.*

In majority voting, all k neighbors get an equal vote of 1. With the other three shapes, the weight is 1 when the distance is 0 and drops to the value of a parameter w_k when the distance is D_k, which is the distance to the kth nearest neighbor.

The amount of voting weight w_j for the jth neighbor is

$$w_j(D_j, D_k, w_k, shape) = \begin{cases} 1 & \text{if } shape = \text{majority} \\ w_k + \dfrac{(1-w_k)(D_k - D_j)}{D_k} & \text{if } shape = \text{linear} \\ w_k^{D_j^2 / D_k^2} & \text{if } shape = \text{gaussian} \\ w_k^{D_j / D_k} & \text{if } shape = \text{exponential} \end{cases} \tag{11}$$

where w_k is the parameter that determines how much weight the kth neighbor receives; D_j is the distance of the jth nearest neighbor; D_k is the distance to the kth neighbor; and *shape* is the parameter that determines which vote-weighting function to use.

Note that the majority voting scheme does not require the w_k parameter. Also note that if $k = 1$ or $w_k = 1$, then all four shapes are equivalent. As D_k approaches 0, the weight in Equation (11) approaches 1, regardless of the shape. Therefore, if the distance D_k is equal to 0, then a weight of 1 is used for the sake of consistency and to avoid dividing by 0. These shapes are illustrated in Figure 4.

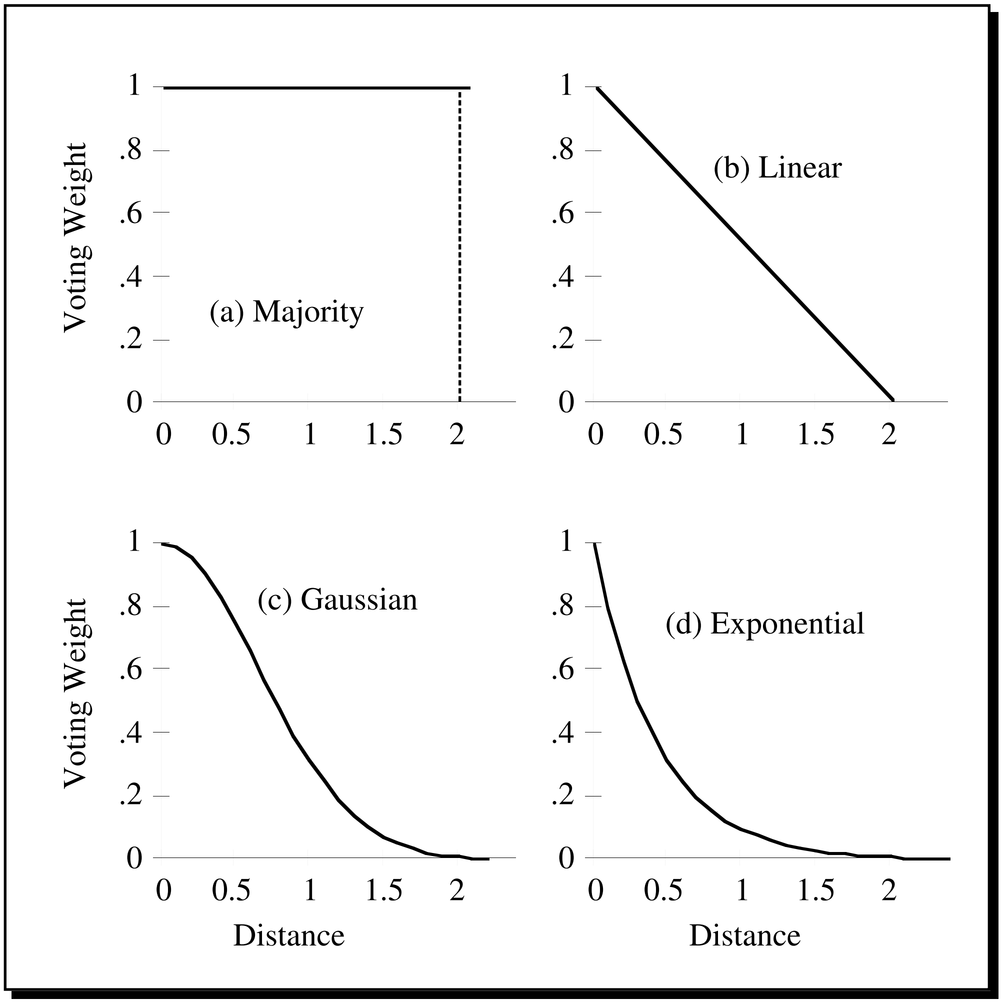


Figure 4. Distance-weighting shapes, shown with $D_k = 2.0$ and $w_k = .01$.

Sometimes it is preferable to use the *average* distance of the k nearest neighbors instead of the distance to the kth neighbor to determine how fast voting weight should drop off. This can be done by computing what the distance D'_k of the kth nearest neighbor would be if the neighbors were distributed evenly. This can be done by setting D'_k to

$$D'_k = \frac{2 \cdot \sum_{i=1}^{k} D_i}{k+1} \tag{12}$$

and using D'_k in place of D_k in Equation 11. When $k = 1$, Equation 12 yields $D'_k = 2D_k / 2 = D_k$, as desired. When $k > 1$, this method can be more robust in the presence of changes in the system such as changing parameters or the removal of instances from the classifier. The flag *avgk* will be used to determine whether to use D'_k instead of D_k.

4.2. Cross-Validation and Confidence (CVC)

Given the distance-weighted voting scheme described in Section 4.1., IDIBL must set the following parameters:

- k, the number of neighbors that vote on the class of a new input vector.
- *shape*, the shape of the distance-weighted voting function.
- *avgk*, the flag determining whether to use the average distance to the k nearest neighbors rather than the kth distance.
- w_k, the weight of the kth neighbor (except in majority voting).

When faced with a choice between two or more sets of parameter values, some method is needed for deciding which is most likely to yield the best generalization. This section describes how these parameters are set automatically, using a combination of leave-one-out cross-validation and confidence levels.

4.2.1. Cross-Validation

With leave-one-out cross-validation (LCV), the generalization accuracy of a model is estimated from the average accuracy attained when classifying each instance i using all the instances in T except i itself. For each instance, the accuracy is 1 if the instance is classified correctly, and 0 if it is misclassified. Thus the average LCV accuracy is r / n, where r is the number classified correctly and n is the number of instances in T. Since r is an integer from 0 to n, there are only $n + 1$ accuracy values possible with this measure, and often two different sets of parameter values will yield the same accuracy because they will classify the same number of instances correctly. This makes it difficult to tell which parameter values to use.

4.2.2. Confidence

An alternative method for estimating generalization accuracy is to use the *confidence* with which each instance is classified. The average confidence over all n instances in the training set can then be used to estimate which set of parameter values will yield better generalization. The confidence for each instance is

$$conf = \frac{votes_{correct}}{\sum_{c=1}^{C} votes_c} \tag{13}$$

where $votes_c$ is the sum of weighted votes received for class c and $votes_{correct}$ is the sum of weighted votes received for the correct class. In terms of fuzzy logic [Zadeh, 1965], the confidence of each instance can also be thought of as its class membership in the correct class.

When majority voting is used, $votes_c$ is simply a count of how many of the k nearest neighbors were of class c, since the weights are all equal to 1. In this case, the confidence will be an integer in the range $0..k$, divided by k, and thus there will be only $k + 1$ possible confidence values for each instance. This means that there will be $(k + 1)(n + 1)$ possible accuracy estimates using confidence instead of $n + 1$ as with LCV, but it is still possible for small changes in parameters to yield no difference in average confidence.

When distance-weighted voting is used, however, each vote is weighted according to its distance and the current set of parameter values, and $votes_c$ is the sum of the weighted votes for each class. Even a small change in the parameters will affect how much weight each neighbor gets and thus will affect the average confidence.

After learning is complete, the confidence can be used to indicate how confident the classifier is in its generalized output. In this case the confidence is the same as defined in Equation 13, except that $votes_{correct}$ is replaced with $votes_{out}$, which is the amount of voting weight received by the class that is chosen to be the output class by the classifier. This is also equal to the maximum number of votes (or maximum sum of voting weights) received by any class, since the majority class is chosen as the output.

4.2.3. Cross-Validation and Confidence (CVC)

Average confidence has the attractive feature that it provides a continuously valued metric for evaluating a set of parameter values. However, it also has drawbacks that make it inappropriate for direct use on the parameters in IDIBL. Average confidence is increased whenever the ratio of votes for the correct class to total votes is increased. Thus, this metric strongly favors $k = 1$ and $w_k = 0$, regardless of their effect on classification, since these settings give the nearest neighbor more relative weight, and the nearest neighbor is of the same class more often than other neighbors. This metric also tends to favor exponential weighting since it drops voting weight more quickly than the other shapes.

Therefore, using confidence as the sole means of deciding between parameter settings will favor any settings that weight nearer neighbors more heavily, even if accuracy is degraded by doing so.

In order to avoid this problem, IDIBL combines cross-validation and confidence into a single metric called *CVC*. Using CVC, the accuracy estimate cvc_i of a single instance i is

$$cvc_i = \frac{n \cdot cv + conf}{n+1} \tag{14}$$

where n is the number of instances in the training set T; $conf$ is as defined in Equation 13; and cv is 1 if instance i is classified correctly by its neighbors in S, or 0 otherwise.

This metric weights the cross-validation accuracy more heavily than the confidence by a factor of n. This technique is equivalent to using $numCorrect + avgConf$ to make decisions, where $numCorrect$ is the number of instances in T correctly classified by their neighbors in S and is an integer in the range $0..n$, and $avgConf$ is the average $conf$ (from Equation 13) for all instances and is a real value in the range 0..1. Thus, the LCV portion of CVC can be thought of as providing the whole part of the score, with confidence providing the fractional part. Dividing $numCorrect + avgConf$ by $n + 1$ results in a score in the range 0..1, as would also be obtained by averaging Equation 14 for all instances in T.

This metric gives LCV the ability to make decisions by itself unless multiple parameter settings are tied, in which case the confidence makes the decision. There will still be a bias towards giving the nearest neighbor more weight, but only when LCV cannot determine which parameter settings yield better leave-one-out accuracy.

4.3. Parameter Tuning

This section describes the learning algorithm used by IDIBL to find the parameters k, w_k, $avgk$, and $shape$, as described in Sections 4.1 and 4.2. The parameter-tuning algorithm assumes that for each instance i in T, the nearest $maxk$ neighbors in S have been found. Parameter tuning takes place both before and after pruning is done. S is equal to T prior to pruning, and S is a subset of T after pruning has taken place.

The neighbors of each instance i, notated as $i.n_1...i.n_{maxk}$, are stored in a list ordered from nearest to furthest for each instance, so that $i.n_1$ is the nearest neighbor of i and $i.n_k$ is the kth nearest neighbor. The distance $i.D_j$ to each of instance i's neighbor $i.n_j$ is also stored to avoid continuously recomputing this distance.

In our experiments, $maxk$ was set to 30 before pruning to find an initial value of k. After pruning, $maxk$ was set to this initial value of k since increasing the size of k does not make much sense after instances have been removed. In addition, the pruning process leaves the list of k (but not $maxk$) neighbors intact, so this strategy avoids the lengthy search to find every instance's nearest neighbors again. In our experiments IDIBL rarely if ever chose a value of k greater than 10, but we used $maxk = 30$ to leave a wide margin of error since not much time was required to test each value of k.

CVC is used by IDIBL to automatically find good values for the parameters k, w_k, $shape$, and $avgk$. Note that none of these parameters affect the distance between neighbors but only the amount of voting weight each neighbor gets. Thus, changes in these parameters can be made without requiring a new search for nearest neighbors or even an update to the stored distance to each neighbor. This allows a set of parameter values to be evaluated in O(kn) time instead of the O(mn^2) time required by a naive application of leave-one-out cross-validation.

To evaluate a set of parameter values, cvc_i as defined in Equation 13 is computed as follows. For each instance i, the voting weight for each of its k nearest neighbors $i.n_j$ is found according to $w_j(i.D_j, D_k, w_k, shape)$ defined in Equation 11, where d_k is $i.D_k$ if $avgk$ is false, or D_k' as defined in Equation 12 if $avgk$ is true. These weights are summed in their separate respective classes, and the confidence of the correct class is found as in Equation 13. If the majority class is the same as the true output class of instance i, cv in Equation 13 is 1. Otherwise, it is 0. The average value of cvc_i over all n instances is used to determine the fitness of the parameter values.

The search for parameter values proceeds in a greedy manner as follows. For each iteration, one of the four parameters is chosen for adjustment, with the restriction that no parameter can be chosen twice in a row, since doing so would simply rediscover the same parameter value. The chosen parameter is set to various values as explained below while the remaining parameters are held constant. For each setting of the chosen parameter, the CVC fitness for the system is calculated, and the value that achieves the highest fitness is chosen as the new value for the parameter.

At that point, another iteration begins, in which a different parameter is chosen at random and the process is repeated until several attempts at tuning parameters does not improve the best CVC fitness found so far. In practice, only a few iterations are required to find good settings, after which improvements cease and the search soon terminates. The set of parameters that yield the best CVC fitness found at any point during the search are used by IDIBL for classification. The four parameters are tuned as follows.

1. Choosing *k*. To pick a value of k, all values from 2 to *maxk* (=30 in our experiments) are tried, and the one that results in maximum CVC fitness is chosen. Using the value $k = 1$ would make all of the other parameters irrelevant, thus preventing the system from tuning them, so only values 2 through 30 are used until all iterations are complete.

2. Choosing a shape function. Picking a vote-weighting shape function proceeds in a similar manner. The shapes *linear*, *gaussian*, and *exponential* are tried, and the shape that yields the highest CVC fitness is chosen. Using *majority* voting would make the parameters *wk* and *avgk* irrelevant, so this setting is not used until all iterations are complete. At that point, majority voting is tried with values of k from 1 to 30 to test both $k = 1$ and majority voting in general, to see if either can improve upon the tuned set of parameters.

3. Setting *avgk*. Selecting a value for the flag *avgk* consists of simply trying both settings, i.e., using D_k and D'_k and seeing which yields higher CVC fitness.

4. Searching for w_k. Finding a value for w_k is more complicated because it is a real-valued parameter. The search for a good value of w_k begins by dividing the range 0..1 into ten subdivisions and trying all eleven endpoints of these divisions. For example, for the first pass, the values 0, .1, .2, ..., .9, and 1.0 are used. The value that yields the highest CVC fitness is chosen, and the range is narrowed to cover just one division on either side of the chosen value, with the constraint that the range cannot go outside of the range 0..1. For example, if .3 is chosen in the first round, then the new range is from .2 to .4. The process is repeated three times, at which point the effect on classification becomes negligible.

IDIBL tunes each parameter separately in a random order until several attempts at tuning parameters does not improve the best CVC fitness found so far. After each pass, if the CVC fitness is the best found so far, the current parameter settings are saved. The parameters that resulted in the best fitness during the entire search are then used.

Tuning each parameter takes $O(kn)$ time, so the entire process takes $O(knt)$ time, where t is the number of iterations required before the stopping criterion is met. In practice t is small (e.g., less than 20), since tuning each parameter once or twice is usually sufficient. These time requirements are quite acceptable, especially compared to algorithms that require repeated $O(mn^2)$ steps.

Pseudo-code for the parameter-finding portion of the learning algorithm is shown in Figure 5. This algorithm assumes that the nearest *maxk* neighbors of each instance T have been found and returns the parameters that produce the highest CVC fitness of any tried. Once these parameters have been found, the neighbor lists can be discarded, and only the raw instances and best parameters need to be retained for use during subsequent classification.

In Figure 5, to “try” a parameter value means to set the parameter to that value, find the CVC fitness of the system, and, if the fitness is better than any seen so far, set *bestCVC* to this fitness, and remember the current set of parameter values in *bestParams*.

```
FindParams(maxTime, training set T): bestParams
    Assume that the maxk nearest neighbors have been
        found for each instance i in T.
    Let timeSinceImprovement=0.
    Let bestCVC=0.
    While timeSinceImprovement < maxTime
        Choose a random parameter p to adjust.
        If (p="k") try k=2..30, and set k to best value found.
        If (p="shape") try linear, gaussian, and exponential.
        If (p="avgk") try D_k and D'_k.
        If (p="w_k")
            Let min=0 and max=1
            For iteration=1 to 3
                Let width=(min-max)/10.
                Try w_k=min..max in steps of width.
                Let min=best w_k-width (if min<0, let min=0)
                Let max=best w_k+width (if max>1, let max=1)
            Endfor
        If bestCVC was improved during this iteration,
            then let timeSinceImprovement=0,
            and let bestParams=current parameter settings.
    Endwhile.
    Let shape=majority, and try k=1..30.
    if bestCVC was improved during this search,
        then let bestParams=current parameter settings.
    Return bestParams.
```

Figure 5. Pseudo-code for parameter-finding algorithm.

In experiments on 31 datasets from the UCI machine learning database repository, an algorithm using the CVC method to tune parameters achieved higher average accuracy than the same algorithm using only cross-validation to tune the parameters, which in turn had higher accuracy than an identical algorithm that used majority voting [Wilson & Martinez, 1997d].

5. IDIBL Learning Algorithm

Sections 2-4 present several independent extensions that can be applied to instance-based learning algorithms. This section shows how these pieces fit together in the *Integrated Decremental Instance-Based Learning* (IDIBL) algorithm. The learning algorithm proceeds according to the following five steps.

Step 1. Find IVDM Probabilities. IDIBL begins by calling *FindProbabilities* as described in Section 2.3 and outlined in Figure 1. This builds the probability values needed by the IVDM distance function. This distance function is used in all subsequent steps. This step takes O(mn) time, where n is the number of instances in the training set and m is the number of input attributes in the domain.

Step 2. Find Neighbors. IDIBL then finds the nearest *maxk* (=30 in our implementation) neighbors of each instance i in the training set T. These neighbors are stored in a list as described in Section 4, along with their distances to i, such that the nearest neighbor is at the head of the list. This step takes $O(mn^2)$ time and is typically the most time-intensive part of the algorithm.

Step 3. Tune Parameters. Once the neighbors are found, IDIBL initializes the parameters with some default values ($w_k = 0.2$, *shape* = *linear*, *avgk* = *true*, $k = 3$) and calls *FindParameters*, with $S = T$, as described in Section 4.3 and outlined in Figure 5. It actually forces the first iteration to tune the parameter k, since that parameter can have such a large effect on the behavior of the others. It continues until four iterations yield no further improvement in CVC fitness, at which point each of the four parameters has had a fairly good chance of being tuned without yielding improvement. This step takes $O(kn)$ time.

Step 4. Prune the Instance Set. At that point, DROP4 is called in order to find a subset S of T to use in subsequent classification. DROP4 uses the best parameters found in the previous step in all of its operations. This step takes $O(mn^2)$ time, though in practice it is several times faster than Step 2, since the $O(mn)$ step must only be done when an instance is pruned, rather than for every instance, and only the instances in S must be searched when the $O(mn)$ step is required.

Step 5. Retune Parameters. Finally, IDIBL calls *FindParameters* one more time, except that this time all of the instances in T have neighbors only in S. Also, *maxk* is set to the value of k found in step 4 instead of the original larger value. *FindParameters* continues until eight iterations yield no further improvement in CVC fitness. Step 3 is used to get the parameters in a generally good area so that pruning will work properly, but Step 5 tunes the parameters for more iterations before giving up in order to find the best set of parameters reasonably possible.

```
LearnIDIBL(training set T):S, bestParams, P_a,v,c
    P_a,v,c = FindProbabilities(T)
    For each instance i in T
        Find nearest maxk (=30) neighbors of instance i.
    bestParams = FindParameters(4,T)
    S = DROP4(T)
    Let maxk = k from bestParams
    bestParams = FindParameters(8,T)
    Return S, bestParams, and P_a,v,c
```

Figure 6. Pseudo-code for the IDIBL learning algorithm.

High-level pseudo-code for the IDIBL learning algorithm is given in Figure 6, using pseudo-code for *FindProbabilities* from Figure 1, *DROP4* from Figure 3, and *FindParameters* from Figure 5.

At this point, IDIBL is ready to classify new input vectors it has not seen before. The lists of neighbors maintained by each instance can be disposed of, as can all pruned instances (unless later updates to the model are anticipated). Only the subset S of instances, the four tuned parameters, and the probability values $P_{a,v,c}$ required by IVDM need be retained.

When a new input vector $\boldsymbol{y}$ is presented to IDIBL for classification, IDIBL finds the distance between $\boldsymbol{y}$ and each instance in S using IVDM. The nearest k neighbors (using the best value of k found in Step 5) vote using the other tuned parameters in the distance-weighted voting scheme, and the output class that has the highest confidence is used as the output class of $\boldsymbol{y}$. The confidence achieved indicates how confident IDIBL is in its classification.

The learning algorithm is dominated by the $O(mn^2)$ step required to build the list of nearest neighbors for each instance, where n is the number of instances in T and m is the number of input attributes. Classifying an input vector takes only $O(rm)$ time, where r is the number of instances in the reduced set S, compared to the $O(nm)$ time required by the basic nearest neighbor algorithm. This learning step is done just once, while subsequent classification is done indefinitely, so the time complexity of the algorithm is less than that of the nearest neighbor rule in the long run.

6. Empirical Results

IDIBL was implemented and tested on 44 datasets from the UCI machine learning database repository.

Each test consists of ten trials, each using one of ten partitions of the data randomly selected from the data sets, i.e., 10-fold cross-validation. For each trial, 90% of the available training instances were used for the training set T, and the remaining 10% of the instances were classified using only the instances remaining in the subset S. The average generalization accuracy over the ten trials is reported for each test.

In order to see how the extensions in IDIBL affect generalization accuracy, results for several instance-based learning algorithms are given for comparison. The *k*NN algorithm is a basic *k*-nearest neighbor algorithm that uses k=3 and majority voting. The *k*NN algorithm uses normalized Euclidean distance for linear attributes and the overlap metric for nominal attributes.

The *IVDM* column gives results for a *k*NN classifier that uses IVDM as the distance function instead of the Euclidean/overlap metric. The *DROP4* column shows the accuracy when the *k*NN classifier using the IVDM distance function is pruned using the DROP4 reduction technique.

Finally, the column labeled *IDIBL* adds the distance-weighting and parameter-tuning steps. The "(size)" column appearing after the DROP4 and IDIBL columns indicate what percentage of the instances were retained for use during classification for each dataset. For the *k*NN and IVDM columns, 100% of the instances in the training set are used for classification.

As can be seen from Table 1, each enhancement raises the average generalization accuracy on these datasets, including DROP4, which reduces storage from 100% to 14.14%. Though these datasets are not particularly noisy, in experiments where noise was added to the output class in the training set, DROP4 was more noise-tolerant than the unpruned system due to its noise-filtering pass [Wilson & Martinez, 1997c].

The IDIBL system achieves the highest accuracy of any of the methods, indicating that each enhancement yields an improvement that is independent of the others. It does sacrifice some degree of storage reduction when compared to the DROP4 column. When distance-weighted voting is used and parameters are tuned to improve CVC fitness, a more precise decision boundary is found that may not allow for quite as many border instances to be removed. In addition, the parameter-tuning step can choose values for k larger the default value of $k = 3$ used by the other algorithms, which can also prevent DROP4 in IDIBL from removing as many instances.

Dataset	kNN	IVDM	DROP4	(size)	IDIBL	(size)
Annealing	94.61	96.11	94.49	9.30	95.96	7.67
Audiology	72.00	77.50	70.87	25.81	72.14	22.86
Australian	81.16	80.58	85.37	7.59	85.36	11.60
Breast Cancer (WI)	95.28	95.57	96.28	3.85	97.00	5.63
Bridges	53.73	60.55	59.55	24.00	63.18	34.89
Credit Screening	81.01	80.14	85.94	6.96	85.35	11.29
Echocardiogram	94.82	100.00	100.00	13.07	100.00	9.91
Flag	48.84	57.66	61.29	21.82	57.66	32.07
Glass	70.52	70.54	69.59	25.49	70.56	38.68
Heart Disease	75.56	81.85	83.71	15.43	83.34	24.28
Heart (Cleveland)	74.96	78.90	80.81	15.44	83.83	29.37
Heart (Hungarian)	74.47	80.98	80.90	12.96	83.29	18.06
Heart (Long-Beach-VA)	71.00	66.00	72.00	7.11	74.50	14.78
Heart (More)	71.90	73.33	76.57	14.56	78.39	20.46
Heart (Swiss)	91.86	87.88	93.46	2.44	93.46	5.78
Hepatitis	77.50	82.58	79.29	12.83	81.88	18.43
Horse-Colic	60.82	76.78	76.46	19.79	73.80	26.73
Image Segmentation	93.57	92.86	93.81	10.87	94.29	15.50
Ionosphere	86.33	91.17	90.88	6.52	87.76	21.18
Iris	95.33	94.67	95.33	8.30	96.00	10.15
LED+17 noise	42.90	60.70	70.50	15.20	73.60	40.09
LED	57.20	56.40	72.10	12.83	74.88	43.89
Liver Disorders	63.47	58.23	63.27	27.44	62.93	43.92
Monks-1	69.43	68.09	85.42	19.45	87.74	23.20
Monks-2	54.65	97.50	82.06	34.49	87.15	36.32
Monks-3	78.49	100.00	100.00	3.86	100.00	2.60
Pima Indians Diabetes	70.31	69.28	72.40	18.29	75.79	29.28
Promoters	82.09	92.36	85.91	18.34	88.64	21.80
Sonar	86.60	84.17	81.64	23.56	84.12	50.10
Soybean (Large)	89.20	92.18	86.29	27.32	87.60	31.03
Soybean (Small)	100.00	100.00	100.00	28.37	100.00	18.92
Thyroid (Allbp)	94.89	95.32	96.00	2.58	96.11	2.89
Thyroid (Allhyper)	97.00	97.86	97.50	2.53	97.10	3.51
Thyroid (Allhypo)	90.39	96.07	95.50	4.63	95.71	5.31
Thyroid (Allrep)	96.14	98.43	97.82	2.32	98.29	3.07
Thyroid (Dis)	98.21	98.04	97.71	2.41	97.97	2.53
Thyroid (Hypothyroid)	93.42	98.07	98.42	1.87	98.45	2.34
Thyroid (Sick)	86.89	96.86	96.68	2.71	96.75	3.11
Thyroid (Sick-Euthyroid)	68.23	95.07	95.38	4.74	95.08	6.08
Vehicle	70.22	69.27	68.57	25.86	72.62	37.31
Voting	93.12	95.17	96.08	6.05	95.62	11.34
Vowel	98.86	97.53	86.92	41.65	90.53	33.57
Wine	95.46	97.78	95.46	9.92	93.76	8.74
Zoo	94.44	98.89	92.22	21.61	92.22	22.22
Average:	**80.38**	**84.98**	**85.46**	14.14	**86.37**	19.60

Table 1. Generalization accuracy of a basic *k*NN classifier, one enhanced with IVDM, IVDM enhanced with DROP4, and the full IDIBL system.

In order to see how IDIBL compares with other popular machine learning models, the results of running IDIBL on 35 datasets were compared with results reported by Zarndt [1995]. Zarndt's results are also based on 10-fold cross-validation. He reported results for 16 learning algorithms, from which we have selected one representative learning algorithm from each general class of algorithms. Where more than one algorithm was available in a class (e.g., several decision tree models were available), the one that achieved the highest results in Zarndt's experiments is reported here.

Results for IDIBL are compared to those achieved by the following algorithms:

- C4.5 [Quinlan, 1993], an inductive decision tree algorithm. Zarndt also reported results for ID3 [Quinlan, 1986], C4, C4.5 using induced rules [Quinlan, 1993], Cart [Breiman et al., 1984], and two decision tree algorithms using minimum message length [Buntine, 1992].
- CN2 (using ordered lists) [Clark & Niblett, 1989], which combines aspects of the AQ rule-inducing algorithm [Michalski, 1969] and the ID3 decision tree algorithm [Quinlan, 1986]. Zarndt also reported results for CN2 using unordered lists [Clark & Niblett, 1989].
- a "naive" Bayesian classifier [Langley, Iba & Thompson, 1992; Michie, Spiegelhalter & Taylor, 1994] (*Bayes*).
- the Perceptron [Rosenblatt, 1959] single-layer neural network (*Per*).
- the Backpropagation [Rumelhart & McClelland, 1986] neural network (*BP*).
- IB1-4 [Aha, Kibler & Albert, 1991; Aha, 1992], four instance-based learning algorithms.

All of the instance-based models reported by Zarndt were included since they are most similar to IDIBL. IB1 is a simple nearest neighbor classifier with k=1. IB2 prunes the training set, and IB3 extends IB2 to be more robust in the presence of noise. IB4 extends IB3 to handle irrelevant attributes. All four use the Euclidean/overlap metric.

The results of these comparisons are presented in Table 2. The highest accuracy achieved for each dataset is shown in bold type. The average over all datasets is shown in the bottom row. We exclude results for several of the datasets that appear in Table 1 for which the results are not directly comparable.

As can be seen from the results in the table, no algorithm had the highest accuracy on all of the datasets, due to the *selective superiority* [Brodley, 1993] of each algorithm, i.e., the degree to which each bias [Mitchell, 1980] is appropriately matched for each dataset [Dietterich, 1989; Wolpert, 1993; Schaffer, 1994; Wilson & Martinez, 1997e]. However, IDIBL had the highest accuracy of any of the algorithms for more of the datasets than any of the other models and had the highest overall average generalization accuracy.

The results are theoretically limited to this set of applications, and the accuracy for some algorithms might be improved by a more careful tuning of system parameters, but the results indicate that IDIBL is a robust learning system that can be successfully applied to a variety of real-world problems.

Dataset	C4.5	CN2	Bayes	Per	BP	IB1	IB2	IB3	IB4	IDIBL
Annealing	94.5	98.6	92.1	96.3	**99.3**	95.1	96.9	93.4	84.0	96.0
Audiology	77.5	81.1	**85.4**	76.7	81.9	77.5	72.2	63.3	68.2	72.1
Australian	**85.4**	82.0	83.1	84.9	84.5	81.0	74.2	83.2	84.5	**85.4**
Breast Cancer	94.7	95.2	93.6	93.0	96.3	96.3	91.0	95.0	94.1	**97.0**
Bridges	56.5	58.2	66.1	64.0	**67.6**	60.6	51.1	56.8	55.8	63.2
Credit Screening	83.5	83.0	82.2	83.6	85.1	81.3	74.1	82.5	84.2	**85.4**
Echocardiogram	90.1	83.2	90.0	87.0	89.3	84.0	80.2	83.9	80.2	**100**
Flag	56.2	51.6	52.5	45.3	**58.2**	56.6	52.5	53.1	53.7	57.7
Glass	65.8	59.8	**71.8**	56.4	68.7	71.1	67.7	61.8	64.0	70.6
Heart Disease	73.4	78.2	75.6	80.8	82.6	79.6	73.7	72.6	75.9	**83.3**
Hepatitis	55.7	63.3	57.5	67.3	68.5	66.6	65.8	63.5	54.1	**81.9**
Horse-Colic	70.0	65.1	68.6	60.1	66.9	64.8	59.9	56.1	62.0	**73.8**
Ionosphere	90.9	82.6	85.5	82.0	**92.0**	86.3	84.9	85.8	89.5	87.8
Iris	94.0	92.7	94.7	95.3	96.0	95.3	92.7	95.3	**96.6**	96.0
LED+17 noise	66.5	61.0	64.5	60.5	62.0	43.5	39.5	39.5	64.0	**73.6**
LED	70.0	68.5	68.5	70.0	69.0	68.5	63.5	68.5	68.0	**74.9**
Liver Disorders	62.6	58.0	64.6	66.4	**69.0**	62.3	61.7	53.6	61.4	62.9
Monks-1	98.9	**100**	90.5	72.3	**100**	99.5	87.5	71.4	76.1	87.7
Monks-2	64.5	86.5	79.0	61.6	**100**	76.2	67.4	57.9	55.6	87.2
Monks-3	98.9	97.7	98.7	98.2	98.7	96.6	90.6	89.9	89.7	**100**
Pima Indians Diabetes	72.7	65.1	72.2	74.6	75.8	70.4	63.9	71.7	70.6	**75.8**
Promoters	77.3	87.8	78.2	75.9	87.9	82.1	72.3	77.2	79.2	**88.6**
Sonar	73.0	55.4	73.1	73.2	76.4	**86.5**	85.0	71.1	71.1	84.1
Soybean (Small)	98.0	**100**	**100**	**100**	**100**	**100**	**100**	97.8	97.8	**100**
Thyroid (Allbp)	97.3	95.5	**97.4**	96.3	96.7	96.2	93.8	91.3	90.0	96.1
Thyroid (Allhyper)	**98.9**	97.3	98.8	97.9	98.1	97.5	95.5	93.9	94.1	97.1
Thyroid (Allhypo)	**99.5**	93.8	**99.5**	93.7	93.7	91.4	84.7	79.9	79.4	95.7
Thyroid (Allrep)	**99.2**	96.5	**99.2**	96.6	97.5	96.5	95.0	90.7	90.2	98.3
Thyroid (Dis)	**98.9**	98.3	98.8	97.9	98.6	98.1	96.0	90.6	92.9	98.0
Thyroid (Hypothyroid)	99.2	95.2	**99.3**	98.0	97.5	97.0	92.0	89.9	90.3	98.5
Thyroid (Sick)	98.8	93.6	**99.0**	96.6	95.3	95.8	93.2	91.8	90.3	96.8
Thyroid (Sick-Euthyroid)	**97.8**	90.5	97.5	93.8	93.1	92.8	88.4	88.9	86.6	95.1
Voting	96.8	93.8	**95.9**	94.5	95.0	92.4	91.2	90.6	92.4	95.6
Wine	93.3	90.9	94.4	**98.3**	98.3	94.9	93.2	91.5	92.7	93.8
Zoo	93.3	96.7	**97.8**	96.7	95.6	96.7	95.6	94.5	91.1	92.2
Average:	**84.1**	**82.8**	**84.7**	**82.4**	**86.7**	**83.7**	**79.6**	**78.2**	**79.2**	**86.9**

Table 2. Generalization accuracy of IDIBL and several well-known machine learning models.

7. Conclusions and Future Research

The basic nearest neighbor algorithm has had success in some domains but suffers from inadequate distance functions, large storage requirements, slow execution speed, a sensitivity to noise, and an inability to fine-tune its concept description.

The *Integrated Decremental Instance-Based Learning* (IDIBL) algorithm combines solutions to each of these problems into a comprehensive learning system. IDIBL uses the *Interpolated Value Difference Metric* (IVDM) to provide an appropriate distance measure between input vectors that can have both linear and nominal attributes. It uses the *DROP4* reduction technique to reduce storage requirements, improve classification speed, and reduce sensitivity to noise. It also uses a distance-weighted voting scheme with parameters that are tuned using a combination of cross-validation accuracy and confidence in order to provide a more flexible concept description.

In experiments on 44 datasets, IDIBL improved upon the generalization accuracy of similar algorithms that did not include all of the enhancements. When compared with results reported for other popular learning algorithms, IDIBL achieved higher average generalization accuracy than any of the others.

Since each algorithm is better suited for some problems than others, a key area of future research is to understand under what conditions each algorithm—including IDIBL—is successful, so that an appropriate algorithm can be chosen for particular applications, thus increasing the chance of achieving high generalization accuracy in practice.

References

Aha, David W., and Robert L. Goldstone, (1992). "Concept Learning and Flexible Weighting," in *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, IN: Lawrence Erlbaum, pp. 534-539.

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.

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

Atkeson, Chris, (1989). Using local models to control movement. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2*. San Mateo, CA: Morgan Kaufmann.

Batchelor, Bruce G., (1978). *Pattern Recognition: Ideas in Practice*. New York: Plenum Press, pp. 71-72.

Biberman, Yoram, (1994). A Context Similarity Measure. *In Proceedings of the European Conference on Machine Learning (ECML-94)*. Catalina, Italy: Springer Verlag, pp. 49-63.

Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone, (1984). *Classification and Regression Trees*, Wadsworth International Group, Belmont, CA.

Brodley, Carla E., (1993). "Addressing the Selective Superiority Problem: Automatic Algorithm/Model Class Selection," *Proceedings of the Tenth International Machine Learning Conference,* Amherst, MA, pp. 17-24.

Broomhead, D. S., and D. Lowe (1988). Multi-variable functional interpolation and adaptive networks. *Complex Systems*, vol. 2, pp. 321-355.

Buntine, Wray, (1992). "Learning Classification Trees," *Statistics and Computing*, vol. 2, pp. 63-73.

Cameron-Jones, R. M., (1995). Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing. *In Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence*, pp. 99-106.

Carpenter, Gail A., and Stephen Grossberg, (1987). "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115.

Chang, Chin-Liang, (1974). "Finding Prototypes for Nearest Neighbor Classifiers," *IEEE Transactions on Computers*, vol. 23, no. 11, November 1974, pp. 1179-1184.

Clark, Peter, and Tim Niblett, (1989). "The CN2 Induction Algorithm," *Machine Learning*, vol. 3, pp. 261-283.

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

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.

Dasarathy, Belur V., and Belur V. Sheela, (1979). “A Composite Classifier System Design: Concepts and Methodology,” *Proceedings of the IEEE*, vol. 67, no. 5, May 1979, pp. 708-713.

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

Deng, Kan, and Andrew W. Moore, (1995). “Multiresolution Instance-Based Learning,” to appear in *The Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI’95).*

Diday, Edwin, (1974). Recent Progress in Distance and Similarity Measures in Pattern Recognition. *Second International Joint Conference on Pattern Recognition*, pp. 534-539.

Dietterich, Thomas G., (1989). Limitations on Inductive Learning. In *Proceedings of the Sixth International Conference on Machine Learning*. San Mateo, CA: Morgan Kaufmann, pp. 124-128.

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).*

Dudani, Sahibsingh A., (1976). “The Distance-Weighted *k*-Nearest-Neighbor Rule,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, no. 4, April 1976, pp. 325-327.

Gates, G. W. (1972). “The Reduced Nearest Neighbor Rule,” *IEEE Transactions on Information Theory*, vol. IT-18, no. 3, pp. 431-433.

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.

Hart, P. E., (1968). “The Condensed Nearest Neighbor Rule,” Institute of Electrical and Electronics Engineers Transactions on Information Theory, vol. 14, pp. 515-516.

Hecht-Nielsen, R., (1987). Counterpropagation Networks. *Applied Optics*, vol. 26, no. 23, pp. 4979-4984.

Keller, James M., Michael R. Gray, and James A. Givens, Jr., (1985). “A Fuzzy *K*-Nearest Neighbor Algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, no. 4, July/August 1985, pp. 580-585.

Kohavi, Ron, (1995). “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection,” In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI’95).*

Kohonen, Teuvo, (1990). The Self-Organizing Map. *In Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464-1480.

Langley, Pat, Wayne Iba, and Kevin Thompson, (1992). “An Analysis of Bayesian Classifiers,” In *Proceedings of the 10th National Conference on Artificial Intelligence, (AAAI-92)*, AAAI Press/MIT Press, Cambridge, Massachusetts, pp. 223-228.

Lebowitz, Michael, (1985). “Categorizing Numeric Information for Generalization,” Cognitive Science, vol. 9, pp. 285-308.

Lowe, David G., (1995). “Similarity Metric Learning for a Variable-Kernel Classifier,” to appear in *Neural Computation*., vol. 7, no. 1, pp. 72-85.

Merz, C. J., and P. M. Murphy, (1996). *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: http://www.ics.uci.edu/~mlearn/ MLRepository.html.

Michalski, Ryszard S., (1969). “On the quasi-minimal solution of the general covering problem,” *Proceedings of the Fifth International Symposium on Information Processing*, Bled, Yugoslavia, pp. 12-128.

Michalski, Ryszard S., Robert E. Stepp, and Edwin Diday, (1981). A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts. *Progress in Pattern Recognition*, vol. 1, Laveen N. Kanal and Azriel Rosenfeld (Eds.). New York: North-Holland, pp. 33-56.

Michie, D., D. Spiegelhalter, and C. Taylor, (1994). *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, Hertfordshire, England. Book 19.

Mohri, Takao, and Hidehiko Tanaka, "An Optimal Weighting Criterion of Case Indexing for Both Numeric and Symbolic Attributes. In D. W. Aha (Ed.), Case-Based Reasoning: Papers from the 1994 Workshop, Technical Report WS-94-01. Menlo Park, CA: AIII Press, pp. 123-127.

Moore, Andrew W., and Mary S. Lee, (1993). "Efficient Algorithms for Minimizing Cross Validation Error," In *Machine Learning: Proceedings of the Eleventh International Conference*, Morgan Kaufmann.

Nadler, Morton, and Eric P. Smith, (1993). *Pattern Recognition Engineering*. New York: Wiley, pp. 293-294.

Nosofsky, Robert M., (1986). Attention, Similarity, and the Identification-Categorization Relationship. *Journal of Experimental Psychology: General*, vol. 115, no. 1, pp. 39-57.

Papadimitriou, C. H., and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ.

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.

Quinlan, J. R., (1986). "Induction of Decision Trees", *Machine Learning*, vol. 1, pp. 81-106.

Quinlan, J. R., (1989). "Unknown Attribute Values in Induction," *Proceedings of the 6th International Workshop on Machine Learning*, San Mateo, CA: Morgan Kaufmann, pp. 164-168.

Quinlan, J. R., (1993). *C4.5: Programs for Machine Learning*, San Mateo, CA: Morgan Kaufmann.

Rachlin, John, Simon Kasif, Steven Salzberg, David W. Aha, (1994). "Towards a Better Understanding of Memory-Based and Bayesian Classifiers," in *Proceedings of the Eleventh International Machine Learning Conference,* New Brunswick, NJ: Morgan Kaufmann, pp. 242-250.

Renals, Steve, and Richard Rohwer, (1989). "Phoneme Classification Experiments Using Radial Basis Functions," *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'89)*, vol. 1, pp. 461-467.

Ritter, G. L., H. B. Woodruff, S. R. Lowry, and T. L. Isenhour, (1975). "An Algorithm for a Selective Nearest Neighbor Decision Rule," *IEEE Transactions on Information Theory*, vol. 21, no. 6, November 1975, pp. 665-669.

Rosenblatt, Frank, (1959). *Principles of Neurodynamics*, New York, Spartan Books.

Rumelhart, D. E., and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, 1986.

Salzberg, Steven, (1991). "A Nearest Hyperrectangle Learning Method," *Machine Learning*, vol. 6, pp. 277-309.

Schaffer, Cullen, (1994). A Conservation Law for Generalization Performance. *In Proceedings of the Eleventh International Conference on Machine Learning (ML'94)*, Morgan Kaufmann, 1994.

Schaffer, Cullen, (1993). "Selecting a Classification Method by Cross-Validation," *Machine Learning*, vol. 13, no. 1.

Skalak, David B., (1994). "Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms," *Proceedings of the Eleventh International Conference on Machine Learning (ML'94).*

Sproull, Robert F., (1991). "Refinements to Nearest-Neighbor Searching in *k*-Dimensional Trees," *Algorithmica*, vol. 6, pp. 579-589.

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

Tomek, Ivan, (1976). "An Experiment with the Edited Nearest-Neighbor Rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 6, June 1976, pp. 448-452.

Tversky, Amos, (1977). Features of Similarity. *Psychological Review*, vol. 84, no. 4, pp. 327-352.

Wasserman, Philip D., (1993). *Advanced Methods in Neural Computing*, New York, NY: Van Nostrand Reinhold, pp. 147-176.

Watson, I., and F. Marir, (1994). "Case-Based Reasoning: A Review," *The Knowledge Engineering Review*, vol. 9, no. 4.

Wess, Stefan, Klaus-Dieter Althoff and Guido Derwand, (1994). "Using *k*-d Trees to Improve the Retrieval Step in Case-Based Reasoning," Stefan Wess, Klaus-Dieter Althoff, & M. M. Richter (Eds.), *Topics in Case-Based Reasoning*. Berlin: Springer-Verlag, pp. 167-181.

Wettschereck, Dietrich, (1994). "A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm", *To appear in the Proceedings of the 7th European Conference on Machine Learning*.

Wettschereck, Dietrich, and Thomas G. Dietterich, (1995). "An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms," *Machine Learning*, vol. 19, no. 1, pp. 5-28.

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.

Wilson, D. Randall, and Tony R. Martinez, (1996). "Instance-Based Learning with Genetically Derived Attribute Weights," *International Conference on Artificial Intelligence, Expert Systems and Neural Networks (AIE'96),* pp. 11-14.

Wilson, D. Randall, and Tony R. Martinez, (1997a). "Improved Heterogeneous Distance Functions," *Journal of Artificial Intelligence Research*, vol. 6, no. 1, pp. 1-34.

Wilson, D. Randall, and Tony R. Martinez, (1997b). "Instance Pruning Techniques," To appear in Fisher, D., ed., *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, Morgan Kaufmann Publishers, San Francisco, CA.

Wilson, D. Randall, and Tony R. Martinez, (1997c). "Reduction Techniques for Exemplar-Based Learning Algorithms," submitted to *Machine Learning Journal*.

Wilson, D. Randall, and Tony R. Martinez, (1997d). "Distance-Weighting and Confidence in Instance-Based Learning," submitted to *Computational Intelligence*, 1997.

Wilson, D. Randall, and Tony R. Martinez, (1997e). "Bias and the Probability of Generalization," submitted to *International Conference on Intelligent Information Systems (IIS'97).*

Wilson, Dennis L., (1972). "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, July 1972, pp. 408-421.

Wolpert, David H., (1993). On Overfitting Avoidance as Bias. Technical Report SFI TR 92-03-5001. Santa Fe, NM: The Santa Fe Institute.

Zadeh, Lotfi A., (1965). "Fuzzy Sets," *Information and Control*, vol. 8, pp. 338-353.

Zarndt, Frederick, (1995). *A Comprehensive Case Study: An Examination of Connectionist and Machine Learning Algorithms*, Master's Thesis, Brigham Young University.

Zhang, Jianping, (1992). "Selecting Typical Instances in Instance-Based Learning," *Proceedings of the Ninth International Conference on Machine Learning*.

Chapter 11

Conclusions & Future Research Directions

"Whew!"—D. R. Wilson

This dissertation began with the goal of taking the basic nearest neighbor algorithm, which had already been successful on a variety of applications, and overcoming its weaknesses in order to create a flexible, robust learning model. This chapter summarizes how the dissertation succeeded in this goal and outlines future research directions.

1. Summary and Contributions

Chapter 1 identifies several weaknesses of the basic nearest neighbor algorithm, including the following:

- Its distance functions are typically inappropriate for applications with both linear and nominal attributes.
- It has large storage requirements because it stores all available training data in the model.
- It is slow during execution because all of the training instances must be searched in order to classify each new input vector.
- Its accuracy degrades rapidly with the introduction of noise.
- Its accuracy degrades with the introduction of irrelevant attributes.
- It has no ability to adjust its decision boundaries after storing the training data.

After the review in Chapter 2 of related work, the dissertation presented a theoretical basis for the belief that learning algorithms really could be "improved" in practice. It then set about addressing each of the above weaknesses.

Chapters 4 and 5 introduce heterogeneous distance functions that yield significantly improved generalization accuracy on applications that have both continuous and nominal attributes. These distance functions can be used in instance-based learning systems, other learning algorithms with distance functions, as well as a variety of other fields such as statistics and cognitive psychology (see Chapter 5 for details).

Chapters 6 and 7 introduce pruning techniques that reduce the storage requirements of instance-based systems and provide a corresponding increase in classification speed. The most successful reduction methods also reduce the sensitivity of the system to noise. These reduction techniques were applied to instance-based learning systems as well as probabilistic neural networks and introduce concepts that may be applied in related areas as well.

Chapter 8 presents a genetic algorithm used to find attribute weights for an instance-based system in order to avoid the detrimental effects of irrelevant attributes.

Chapter 9 introduces a distance-weighted instance-based learning algorithm that allows for a more flexible concept description. It combines the use of cross-validation and confidence to yield an accuracy estimation metric that allows for more precise fine-tuning of parameters. This accuracy metric can be used to adjust other parameters in instance-based learning systems beyond those used in the systems in this dissertation. The metric can also be used in related systems such as probabilistic neural networks.

Chapter 10 combines the most successful elements of the earlier chapters into a comprehensive system called the *Integrated Decremental Instance-Based Learning* (IDIBL) algorithm that achieves higher generalization accuracy than any of the previous systems. It also achieves higher generalization accuracy in our experiments than that reported for a variety of other popular machine learning and neural network models.

The dissertation was successful in helping to overcome each of the noted weaknesses of the basic nearest neighbor rule. The techniques used to overcome each weakness are in most cases applicable to other instance-based learning systems. In many cases the enhancements can be extended beyond the field of instance-based learning systems into other machine learning algorithms, artificial neural networks, and other fields.

The various enhancements to the basic nearest neighbor rule increase learning time in most cases but yield improved accuracy, reduced storage and/or improved classification time over the original algorithm. The price paid for these improvements is paid just once during learning and the benefits last throughout the life of the classifier.

The learning systems presented in this dissertation all perform *classification*, in which the output value is always a discrete class. However, many of the enhancements presented herein can also be applied to systems that perform *regression*, where the output value is continuously valued.

Most of this dissertation has been either published or submitted for publication, so the contributions of each chapter are distributed in a form such that they are available to a wide audience. Source code for some of the systems is available on-line for use by other researchers.

2. Future Research Directions

The IDIBL algorithm does not include the genetically derived attribute weights as used in Chapter 8. While global attribute weights can be successful at identifying completely irrelevant attributes, they are often not helpful—and sometimes harmful—to generalization accuracy when an application does not have irrelevant attributes.

Some researchers have suggested that global attribute weights are not flexible enough to aid generalization in many cases. Aha & Goldstone [1992] presented a system in which a set of attribute weights for each instance was derived, in addition to a set of global attribute weights. Depending on the distance of an input vector from a particular instance, the weight for each attribute is then interpolated between the local and global attribute weight. Such flexible weighting schemes have potential to further improve accuracy.

As can be seen from the various tables of results presented in this dissertation, no single algorithm performs best on all of the applications in any of our experiments. In order to further improve generalization accuracy, it is important to understand the conditions under which each algorithm fails as well as when it works well.

By identifying areas of strength and weakness among different algorithms, it becomes possible to improve algorithms or choose between them in such a way that generalization accuracy can be further improved.

Future research will focus on how to identify such strengths and weaknesses. It will also focus on developing learning algorithms for local as well as global weighting schemes in order to create learning algorithms that are able to adapt even more effectively to individual problems.

Advances in Instance Based Learning Algorithms

D. Randall Wilson

Department of Computer Science

Ph. D. Degree, August 1997

ABSTRACT

The nearest neighbor algorithm and its derivatives, which are often referred to collectively as *instance-based* learning algorithms, have been successful on a variety of real-world applications. However, in its basic form, the nearest neighbor algorithm suffers from inadequate distance functions, large storage requirements, slow execution speed, a sensitivity to noise and irrelevant attributes, and an inability to adjust its decision surfaces after storing the data. This dissertation presents a collection of papers that seek to overcome each of these disadvantages. The most successful enhancements are combined into a comprehensive system called the *Integrated Decremental Instance-Based Learning* algorithm, which in experiments on 44 applications achieves higher generalization accuracy than other instance-based learning algorithms. It also yields higher generalization accuracy than that reported for 16 major machine learning and neural network models.

COMMITTEE APPROVAL:

Tony R. Martinez, Committee Chairman	Theodore A. Norman, Committee Member
Gordon E. Stokes, Committee Member	J. Kelly Flanagan, Committee Member
Parris K. Egbert, Committee Member	Scott N. Woodfield, Graduate Coordinator

5/29/97

Date