In *Proceedings of the International Conference on Intelligent Information Systems (IIS'97)*, pp. 108-114, 1997.

Bias and the Probability of Generalization

D. Randall Wilson, Tony R. Martinez
randy@axon.cs.byu.edu, martinez@cs.byu.edu
Neural Network and Machine Learning Laboratory, http://axon.cs.byu.edu
Computer Science Department, Brigham Young University Provo, UT 84602, USA

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.

Key words: Bias, generalization, inductive learning, conservation law of generalization.

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.

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.

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.

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

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

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.

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

Table 2. Example of a training set.

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*, **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*, **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.