In Proceedings AAAI-94

Bootstrapping Training-Data Representations for Inductive Learning: A Case Study in Molecular Biology

Haym Hirsh and **Nathalie Japkowicz**
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
hirsh@cs.rutgers.edu, nat@paul.rutgers.edu

Abstract

This paper describes a "bootstrapping" approach to the engineering of appropriate training-data representations for inductive learning. The central idea is to begin with an initial set of human-created features and then generate additional features that have syntactic forms that are similar to the human-engineered features. More specifically, we describe a two-stage process for the engineering of good representations for learning: first, generating by hand (usually in consultation with domain experts) an initial set of features that seem to help learning, and second, "bootstrapping" off of these features by developing and applying operators that generate new features that look syntactically like the expert-based features. Our experiments in the domain of DNA sequence identification show that an initial successful human-engineered representation for data can be expanded in this fashion to yield dramatically improved results for learning.

Introduction

Although most of the best-used inductive learning systems (Rumelhart, Hinton, & Williams 1986; Clark & Boswell 1991; Quinlan 1993) assume that data are represented as feature vectors, the world does not always present problems that directly fit the clean model of inductive learning presented by such systems. In addition to the art of taking a potentially ill-defined problem and formulating it as an inductive learning problem, a learning-system user must decide how data should be represented so that a given learning system applies and yields good results. This often involves high-level questions, such as which aspects of individual cases should be included with each example, as well as low-level questions, such as (for the backpropagation learning algorithm) how to represent data as vectors of real values.

Fortunately, in many situations the domain environment from which data are obtained already provides a natural representation for training data. For example, in vision, images are usually obtained as arrays of brightness values. In chess, positions are represented by 8x8 arrays representing the various squares on the board and what piece (if any) resides in each square (or alternatively, as lists of pieces and on which square each piece resides). In problems concerning DNA—the principal domain considered in this paper—data are naturally represented as strings over a four-letter alphabet.

However, even when the problem domain provides a natural representation for data, there is no guarantee that this representation will be good for learning. Learning will only be successful if the regularities that underlie the data can be discerned by the learning system. For example, Hirsh and Noordewier (1994) have shown that re-expressing DNA-sequence data using higher-level features (defined in terms of the "natural" lower-level representation) can dramatically improve the results of learning. The goal of this paper is to introduce a method for building off such manual feature engineering, to improve the quality of training-data representations. Although our method does not remove the need for human input, it provides a way to bootstrap off initial efforts in a less labor-intensive and "expert-intensive" fashion.

Our approach began with the observation that human-engineered higher-level features often fall into distinct syntactic classes. Our assumption was that such syntactic classes are not accidents, but rather that they often reflect some deeper semantic meaning in the domain. When this is true learning can be improved by using additional features that look syntactically like the human-engineered features in the hope that they, too, will reflect deeper semantic meaning in the domain.

In more detail, our approach operates in two stages. In the first, an initial higher-level representation is constructed manually, usually in consultation with domain experts, with new features defined in terms of the existing low-level features in which the raw data are encoded. In the second, expert-free "bootstrapping" stage, a collection of feature-generation operators is created that is able to generate new features that look syntactically like existing features.

To explore this two-staged approach to generating new features we used the problem of learning rules for predicting promoters in DNA. This domain was particularly well-suited to our efforts, in that we were able build off the previous efforts of Hirsh and Noordewier (1994), which provide a problem where the human-engineering of an initial set of higher-level features has already been performed. We were thus able to focus on the "bootstrapping" stage of our feature-creation process for this problem.

We thank Mick Noordewier, Steve Norton, and Chris Matheus for helpful discussions, and the anonymous reviewers for their useful comments. This work was supported by NSF grant IRI-9209795.

We begin the paper with a general overview of molecular biology and the promoter recognition problem. We then describe the raw and Hirsh-Noordewier representations for training data, followed by details of the application of our bootstrapping approach to this problem and an evaluation of our results. We conclude the paper with a discussion of the generality of our approach and prospects for the future.

The Promoter Learning Problem

Although DNA encodes the genetic information about an organism, understanding the chemical basis for how it serves this role is very difficult. One of the complexities in understanding how the chemical properties of DNA achieve its functionality is that only some parts of DNA—known as *coding regions*—contain genetic information. Other parts, known as *regulatory regions*, regulate the chemical processes that operate on DNA. Other regions serve yet other roles, and in some cases the roles of some regions are not even currently understood.

Very little is known about how to differentiate the various DNA regions. The *promoter recognition problem* has as its goal the recognition of a specific type of regulatory region known as a *promoter sequence*, which signals to the chemical processes acting on DNA where a coding region begins. Identifying promoter sequences is currently a long and difficult laboratory-based process and a reliable non-laboratory-based automated procedure for doing so would prove a valuable tool for future efforts in this field.

This paper focuses on the use of inductive learning to form recognition procedures for promoter sequences. The idea is to take a collection of sequences known to be promoters and a collection known not to be and form a classifier that accurately predicts the presence of promoters in uncharacterized DNA. The promoter data that we use are composed of 47 positive examples and 53 negative ones. These data were obtained from the U.C. Irvine Repository of Machine Learning Datasets, but were modified in response to Norton's (1994) critique of the biological flaws underlying the original formulation of the data: first, the training examples were aligned on the right-most start of transcription; second, sequences that had no start of transcription identified in the original reference were removed.

The particular learning algorithm that we use is the decision tree learner C4.5 (Quinlan 1993), and we report results for both pruned and unpruned trees. For the main part of this paper five different representations are considered. Two of these are the pre-existing raw and Hirsh-Noordewier representations, and the other three result from applying our method. Error rates for each representation are the average of 10 sessions of 10-fold cross-validation (Weiss & Kulikowski 1991). A summary of our results, which are discussed in the coming sections, is given in Table 1 (numbers after each "±" are standard deviations for each of the ten-session averages). The differences in values between adjacent rows are all statistically significant with $p < .01$ except for the transition from row 2 to row 3 in both the unpruned and the pruned cases and for the transition from row 4 to row 5 in the pruned case.

Representation	Unpruned (%)	Pruned (%)
Raw	33.3 ± 3.8	30.5 ± 3.0
+ Hirsh-Noordewier	22.9 ± 3.8	22.0 ± 3.7
+ Bootstrap: Stage 1	21.9 ± 1.5	20.6 ± 2.4
+ Bootstrap: Stage 2	17.6 ± 2.5	17.4 ± 3.0
+ Bootstrap: Stage 3	12.9 ± 3.2	15.2 ± 2.9

Table 1: Error rates for various representations.

Initial Representations

Although the preceding section formulated the promoter-recognition problem as an inductive-learning task, learning cannot proceed until a specific representation for the training examples is chosen and used to describe the data. Moreover, the quality of learning depends on the representation selected. Here we are able to bootstrap from a training-data representation generated by past human-engineering efforts that significantly improves the results of learning (Hirsh & Noordewier 1994). This section begins with a description of the low-level representation for DNA sequences as strings over a four-letter alphabet, followed by the higher-level Hirsh-Noordewier representation.

The "Raw" Training Data Representation

DNA is composed of two chains of chemical "building blocks" known as *nucleotides*. These two chains are linked together at consecutive nucleotide locations to form a winding, ladder-like structure (the so-called DNA *double helix*). Each nucleotide building block is made up of a phosphate group, a sugar, and one of four molecular units: adenine, cytosine, guanine, and thymine, typically represented by the characters "a", "c", "g", and "t". The two chains are complementary in that each adenine on one chain is bound to a thymine on the other, while each cytosine on one chain is bound to a guanine on the other, and thus the sequence of nucleotides in one strand is sufficient for representing the information present in each double-stranded DNA. This leads to a natural representation for DNA, namely a sequential listing of all the nucleotides in the DNA nucleotide chain. Computationally this means DNA can be represented as strings over the four-letter nucleotide alphabet ("a", "c", "g", and "t"), and indeed, this is the representation used for storing sequence information in the on-line nucleic-acid libraries developed by biologists.

Representing DNA as strings also leads to a natural feature-vector representation for learning that has become fairly standard in applications of inductive learning to DNA sequences (*e.g.*, Shavlik, Towell, and Noordewier 1992). Each promoter and non-promoter sequence is truncated to be a string of some uniform length over the four-letter alphabet, and the resulting string is converted into a feature vector of the same length, with each position in the string converted into a single feature. Thus, for example, in this work the initial training data are truncated to be strings of length 51 labeled as positive or negative by whether there is a promoter present in the string. The strings are in turn con-

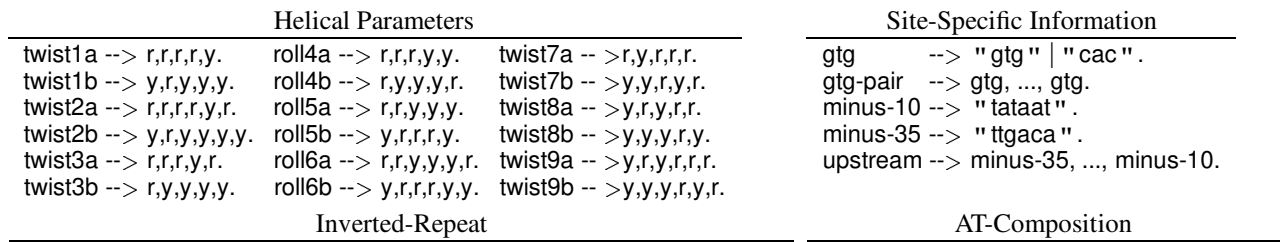


Figure 1: The Hirsh-Noordewier features.

verted into feature vectors of length 51, with each feature taking on one of four values: "a", "c", "g", and "t". Our error rates for learning using this representation are reported in the first row of Table 1.

The Hirsh-Noordewier Representation

The difficulty in using the raw data representation in learning is that the nature of a particular sequence depends on physical and chemical properties of the sequence, properties that are not readily discernible from the raw nucleotide sequence. This observation led to the development by Hirsh and Noordewier (1994) of a set of higher-level features for DNA that reflect such physical and chemical properties in the belief that such a representation would be more successful for learning. Their experimental results show that representing the training data in terms of these higher-level features can lead to dramatic improvements in learning across a range of learners and sequence-identification tasks.

The features we use are listed in Figure 1, represented using an extended form of Prolog's definite clause grammars (Searls 1992; Hirsh & Noordewier 1994). In the twist and roll definitions, "r" stands for "g" or "a" (the *purines*) and "y" stands for "t" or "c" (the *pyrimidines*). Data in this representation are represented as vectors of length 76: the 51 raw features augmented by the 25 features in Figure 1. Our error rates for this representation are reported in the second row of Table 1, and mirror the significantly better learning results reported by Hirsh and Noordewier.[1]

Bootstrapping Better Representations

Although the Hirsh-Noordewier representation significantly improves learning, this work takes their representation further still, using our bootstrapping approach to develop additional successful higher-level features. Doing so required that we create operators that generate new features that look like the Hirsh-Noordewier features. The operators that we created fall into three qualitatively different classes. We therefore applied our bootstrapping approach in three stages, with each corresponding to a different single class of operators. At each stage the new features that we created were monotonically added to the features of the previous stage: no feature, once added to the representation in one stage, is ever removed.

Stage 1: Intra-Feature Extrapolation

Our first stage of bootstrapping begins with the "raw" representation augmented with the Hirsh-Noordewier features, and uses two feature-creation operators to generate new features. These operators were based on patterns observed in the definitions of each of a set of higher-level features (such as the three consecutive "r"s or "y"s at one of the two ends of many of helical-parameter definitions). We call such operators *intra-feature extrapolation* operators, in that they extrapolate from patterns found within the definitions of individual features to new features.

Our first operator extrapolates from patterns in the helical-parameter feature definitions, generating new features that match fairly low-level patterns in these features. In particular, it generates all features of length five or six that satisfy any of the following five patterns observed in the helical-parameter feature definitions in Figure 1:

1. The string begins with exactly 3 or 4 contiguous "r"s.
2. The string ends with exactly 3 or 4 contiguous "y"s.
3. The string has exactly 3 contiguous "r"s in its middle, with the rest "y"s.
4. The string has exactly 3 contiguous "y"s in its middle, with the rest "r"s.
5. The string has exactly 2 contiguous "r"s or "y"s at one of its ends, with the rest alternating.

Our second operator extrapolates from patterns in the site-specific features by switching the nucleotides occurring in feature definitions. It does this by first imposing an artificial generalization hierarchy on the four nucleotide values by dividing them into two groups of two. Each site-specific feature within an existing feature definition is then re-expressed in terms of these groups, and some of the groups are then replaced (consistently across the whole feature definition) to

[1] Note, however, that our error rates are not strictly comparable to those of Hirsh and Noordewier. First, Hirsh and Noordewier use a different collection of promoter data. Second, we use C4.5, whereas their results are for C4.5rules (Quinlan 1993) and backpropagation (Rumelhart, Hinton, & Williams 1986). Finally, our representation differs somewhat from theirs, in that they updated their features after the experiments reported here were completed but before publication of their paper.

be another group. Finally, the artificial group names are reinstantiated to yield a new feature definition. For example, consider the "gtg" and "cac" patterns. We can define the groups G1={g,c} and G2={a,t}, express the two patterns in terms of these groups, "G1 G2 G1", and consider the patterns where G1 and G2 exchange roles, "G2 G1 G2". Replacing the artificial group-names with actual nucleotides yields (in addition to "gtg" and "cac") "gag", "ctc", "aga", "tgt", "aca", and "tct".

By generating all features possible using our first operator we were able to generate 18 new features;[2] a similar application of our second operator in all possible ways gave rise to an additional 15 new features. The resulting representation thus has 109 features, and as shown in Table 1, the new representation gave modest improvements in error-rate estimates, 1.0% for unpruned trees and 1.4% for pruned trees.

Stage 2: Inter-Feature Extrapolation

The next stage of our bootstrapping process extended this representation by applying two additional feature-creation operators that were based on patterns observed *across* feature definitions. For example, each of the five-nucleotide helical-parameter feature definitions is a truncated version of a six-nucleotide helical-parameter definition. We call such operators *inter-feature extrapolation* operators, since they extrapolate from patterns that occur across feature definitions.

Our first inter-feature extrapolation operator is based on the previously mentioned pattern that each helical-parameter feature of length five is derivable from one of length six by deleting an "r" or a "y" either at its start or at its end. This operator takes existing features of length six and deletes the element at its start or its end.

Our second operator is based on an alternative way of viewing the pattern underlying the preceding operator. In particular, each length-six feature can be viewed as the result of adding an "r" or "y" to one of the two ends of a length-five feature. This operator takes feature definitions of length five and creates new features of length six by adding either an "r" or a "y" at the start or end.

These two operators were applied in all possible ways to the features generated by the preceding bootstrapping stage, yielding 43 additional features. As shown in the third row of Table 1, the resulting representation improved learning more significantly than the previous bootstrapping stage, with an additional improvement of 4.3% above the results of the first stage of bootstrapping for unpruned trees, and a 3.2% improvement in the pruned case. Note that these results also lend credibility to the merit of the features generated in the prior bootstrapping stage—the newly created features are based on the Hirsh-Noordewier features *plus* the features created by our first stage of bootstrapping.

[2]Two of these features were actually duplicates. While this does not affect the results of C4.5, it does effect the feature count for the experiments in the next section.

Stage 3: Feature Specialization

Our final bootstrapping stage begins with the preceding bootstrapped representation, and creates new features by specializing existing features. The resulting feature definitions look like existing feature definitions, only they succeed on a smaller number of cases. The single specialization operator we use here simply specializes helical-parameter-type features (the original helical-parameter features plus similar-looking features generated by previous bootstrapping stages) by taking each feature and consistently replacing "r"s with either "a"s or "g"s and "y"s with either "c"s or "t"s. For example, the right-hand side of "twist1a → r,r,r,r,y" would be specialized in four ways: "ggggc", "ggggt", "aaaac", and "aaaat".

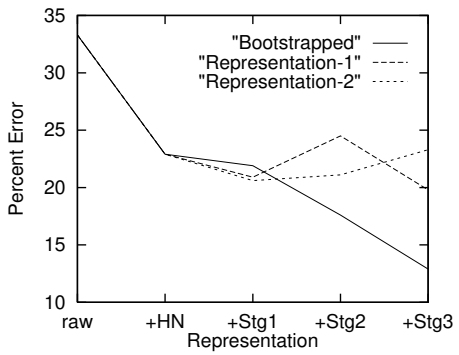
Due to the large number of helical-parameter-type features generated during Stages 1 and 2—each of which would give rise to multiple new features using this operator—it was not practical to apply this operator to all potentially relevant features. We therefore pruned the set of features to which the operator would be applied by selecting only those features that occurred somewhere in the learned decision trees generated by C4.5 on the data using the result of the second stage of bootstrapping. 60 new features were generated by this process, and Table 1 again shows significantly improved error rates: 4.7% from Stage 2 to Stage 3 in the unpruned case, and 2.2% in the pruned case.

Evaluation

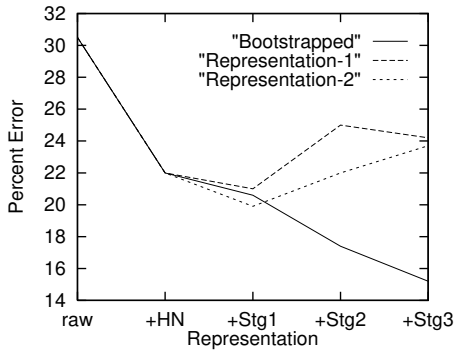
Although the previous section shows that our new features improve learning, they do not tell us what caused the improvement. In particular, we do not know whether similar results would be observed for any collection of new features defined in terms of the raw-data features, as opposed to being due to the new features having syntactic forms similar to existing features. To answer this question we developed four additional representations, each beginning with the raw + Hirsh-Noordewier features, but with additional "random" higher-level features (defined in terms of the raw features) also included in the representation. Each representation contains random features of a single type:

1. Random contiguous strings of length 3 to 10.
2. 2, 3, or 4 random contiguous strings of length 2 to 10 separated by intervals of length 1 to 20.
3. Random contiguous strings of length 5 or 6.
4. 2 or 3 random contiguous strings of length 5 or 6 separated by intervals of length 1 to 40.

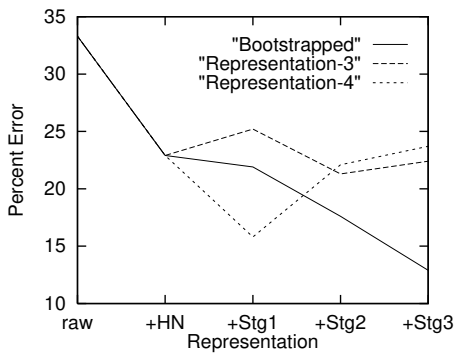
Note that these features are *not* random in the sense that they are "irrelevant features", but rather they are random features defined in terms of the lower-level representation, and as such, they do reflect information about the training data. Since features in the latter two representations are more similar syntactically to the Hirsh-Noordewier features than are features in the first two representations, we report results for the first two separately from the second two, in that the latter constitute a more difficult test for the performance of our newly created features.



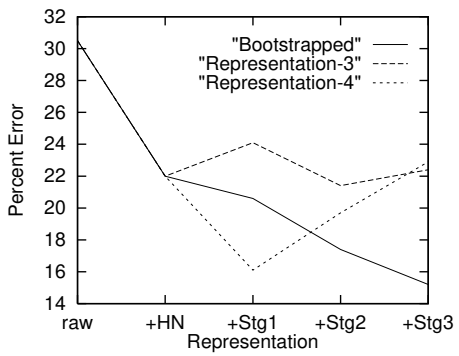
(a) Unpruned C4.5



(b) Pruned C4.5



(c) Unpruned C4.5



(d) Pruned C4.5

Figure 2: Bootstrapped versus random features.

To compare the results of learning using these random representations to the bootstrapped representations, our experiments were conducted to mimic those of the previous section as much as possible. The graphs in Figure 2 all first plot error rates for the raw and raw + Hirsh-Noordewier features. The remaining values are for representations that continue from this point. To get to the first stage of feature addition for each of the random representations (labeled "+Stg1" on the abscissa in each of the graphs) we added 33 random features to equal the same number of features that were added in the first bootstrapping stage above. An additional 43 random features were then added to these 109 features for comparison to the second bootstrapping stage. Finally, an additional 62 features were added to these 152 features to yield the third-stage values.[3] Our results are presented in Figure 2 (where "Bootstrapped" labels the error-rate curve for the series of representations generated in the previous section). Statistical significance was established with $p < .01$ for the differences between the bootstrapped representation and all representations except for the first two random representations for both pruned and unpruned trees at the first bootstrapping stage (Figures 2(a) and 2(b)), and for pruned trees for the final random representation at the second bootstrapping stage (Figure 2(d)).

These results support our claim that the syntactic form of the bootstrapped features plays an important role in their success. Although learning was consistently better with the results of bootstrapping than with the first three random representations, learning with the final random representation was better than bootstrapping when roughly 25 features were added. With more features, however, performance degraded while performance for the bootstrapped features improved. We conjecture that there is some optimal number of such features that can help learning, but if too many such features are used they begin to act truly random, helping the learner find spurious correlations in the data. We leave a more in-depth study of this question for future work. However, given that these features do resemble the syntactic form of the Hirsh-Noordewier features, they can be viewed as additional support that the success of our bootstrapping approach is due the similarity of new features to existing features, rather than simply being the result of adding more features, regardless of syntactic form.

Towards a Bootstrapping Approach to Constructive Induction

Although the feature-creation operators used in our three bootstrapping stages are clearly specific to this domain, we believe there are more general principles underlying them. First, they are based on an intuitively compelling idea, that feature definitions that look like already existing successful feature definitions may themselves also improve learning. Second, intra-feature extrapolation, inter-feature extrapo-

[3] An accurate comparison to the corresponding bootstrapped representation should have involved adding 60 features, rather than 62. Given the large number of features present at this point, we do not believe this error affects our results qualitatively.

lation, and feature specialization (together with additional operations not used in this work, such as feature generalization) are domain-independent notions, even if they manifest themselves here in a domain-dependent fashion.

Further, the implementations of our operators also take a fairly general form, and in particular exhibit two distinct general approaches to encoding feature-creation operators. The first approach includes our initial intra-feature extrapolation operator, and takes the form of a *constraint-based* feature generator. This operator generates all possible features of a given form that satisfy some constraints. The second approach, which includes the remaining operators, generates new features by modifying existing features (such as by deletion of feature-definition elements, specialization, etc.). This latter, *transformational* approach to feature-creation has already been suggested by Callan and Utgoff (1991) and Fawcett and Utgoff (1992). However, unlike this work where transformations are designed so that they generate feature definitions of certain syntactic forms, their transformations are based on background theories of a domain and are generally of a more wide-ranging nature, not driven by notions of syntactic similarity.

In an ideal world a learning system would take the training data and automatically reformulate it into a form better suited for learning. Incorporating such *constructive induction* (Michalski 1983) capabilities into learning systems in a practical way is very difficult. This work represents an initial step in a longer-term goal of developing a domain-independent bootstrapping approach to constructive induction, driven by the afore-mentioned generalities underlying this work. In particular, it is our goal to continue our development of feature-creation operators with an eye towards domain independence. We expect this to require the development of methods for automatically recognizing patterns in a collection of higher-level features and converting them into constraints for use by domain-independent constraint-based feature-generation operators, as well as refining and expanding our current set of transformational operators into a more comprehensive and domain-independent set.

Summary

This paper has presented a two-stage approach to the engineering of good representations for data, using operators that create new features that embody syntactic patterns appearing in an initial set of expert-based features. The assumption underlying this approach is that syntactic patterns commonly underlie successful human-created higher-level features, and moreover, when such patterns are present, that they have semantic meaning in the domain and thus similar-looking features will be more likely to be successful for learning.

The application of this approach to the problem of learning rules for recognizing promoters in DNA sequences generated representations that dramatically improved the results of learning beyond those of the initially successful human-engineered features. Our experiments further show that it is the syntactic similarity of the new features to existing features that is the cause for the success, rather than merely the addition of new features (even when syntactically similar at a more gross level). These results are particularly notable in that both authors of this paper have little training in molecular biology; the bootstrapping operators are a first encoding of syntactic forms that we observed in pre-existing higher-level features, rather than an explicit attempt to encode domain knowledge into the feature-creation operators or an iterative process of using a learning system to find a set of feature-creation operators that generate features that help learning. The ease with which we were able to generate feature-creation operators that significantly improved learning makes us particularly hopeful for the prospects for our longer-term goal of developing this work into a domain-independent bootstrapping approach to constructive induction.

References

Callan, J. P., and Utgoff, P. E. 1991. Constructive induction on domain information. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 614–619. San Mateo, CA: Morgan Kaufmann.

Clark, P. E., and Boswell, R. 1991. Rule induction with CN2. In *Proceedings of the European Working Session on Learning*, 151–163. Berlin: Springer-Verlag.

Fawcett, T. E., and Utgoff, P. E. 1992. Automatic feature generation for problem solving systems. In *Proceedings of the Ninth International Conference on Machine Learning*, 144–153. San Mateo, CA: Morgan Kaufmann.

Hirsh, H., and Noordewier, M. 1994. Using background knowledge to improve learning of DNA sequences. In *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*, 351–357. Los Alamitos, CA: IEEE Computer Society Press.

Michalski, R. S. 1983. A theory and methodology of inductive learning. In Michalski, R. S.; Carbonell, J. G.; and Mitchell, T. M., eds., *Machine Learning: An Artificial Intelligence Approach*. Los Altos, CA: Morgan Kaufmann.

Norton, S. W. 1994. Learning to recognize promoter sequences in E. coli by modeling uncertainty in the training data. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

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

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning internal representations by error propagation. In Rumelhart, D. E., and McClelland, J. L., eds., *Parallel Distributed Processing*. Cambridge, MA: MIT Press. 318–364.

Searls, D. B. 1992. The computational linguistics of biological sequences. In Hunter, L., ed., *Artificial Intelligence and Molecular Biology*. Menlo Park, CA: AAAI Press. 47–120.

Shavlik, J.; Towell, G.; and Noordewier, M. 1992. Using artificial neural networks to refine existing biological knowledge. *International Journal of Human Genome Research* 1:81–107.

Weiss, S. M., and Kulikowski, C. A. 1991. *Computer Systems That Learn*. San Mateo, CA: Morgan Kaufmann.