Machine Learning Approaches to Gene Recognition

Mark W. Craven and Jude W. Shavlik, University of Wisconsin

AS LABORATORIES AROUND THE WORLD PRODUCE EVER-GREATER VOLUMES OF DNA SEQUENCE DATA, EFFICIENT COMPUTATIONAL ANALYSIS TECHNIQUES ARE BECOMING ESSENTIAL. THIS ARTICLE SURVEYS SEVERAL EFFORTS THAT APPLY MACHINE LEARNING TECHNIQUES TO GENE RECOGNITION.

BIOLOGISTS ARE TAKING THE first steps toward knowing the functions and locations of all the genes and regulatory sites in the genomes of several organisms. As these researchers determine the nucleotide sequence of large stretches of human and other DNA, they are producing great volumes of sequence data. Direct laboratory analysis of this data is difficult and expensive, making computational techniques essential. But the variation, complexity, and incompletely understood nature of genes make it impractical to hand-code the algorithms.

Several researchers are exploring how to apply machine learning techniques to gene recognition. Machine learning methods are well suited to sequence analysis because they can learn useful descriptions of genetic concepts when given only instances, rather than explicit definitions, of those concepts. This article looks at several such approaches to gene recognition in two broad classes: *search by signal* and *search by content*. (For the uninitiated in either field, two sidebars offer some background in molecular biology and the type of machine learning known as *empirical learning*.)

Search by signal

Search by signal locates genes indirectly by finding signals (localized regions of DNA with specific functions) that are associated with gene expression. The approaches we describe here formulate their task as classification: They view a DNA sequence through a fixed-length "window" to see if the signal of interest occupies a particular position in the window. In Figure 1, for example, a positive example occurs when a promoter begins at position 3.

Several signals are especially germane to identifying genes:

- transcription initiation sites (*promoters*),
- transcription termination sites (*terminators*),
- translation initiation sites (*start codons*),
- translation termination sites (*stop codons*), and
- splice junction sites

The difficulty of identifying these sites varies considerably. Translation termination sites are trivial to identify: We only need to find a stop codon in the reading frame of a coding region. However, identifying the other types of sites is more complex.

Translation initiation sites. A ribosome does not begin to translate mRNA to protein with its first nucleotide triplet, but rather somewhere downstream, usually with the codon A-U-G, which encodes the amino acid methionine. However, the first A-U-G codon is not necessarily the start codon, and A-U-G may also occur in the middle of a coding region.

The problem is even more difficult for prokaryotic organisms, where a single mRNA molecule may have several translation initiation sites because consecutive genes may be transcribed into a single mRNA chain. Furthermore, translation in prokaryotes sometimes begins with codons other than A-U-G. Fortunately, a so-called *Shine-Dalgarno region* — a sequence that is complementary to the part of the ribosome that binds to mRNA — usually precedes a

0885-9000/94/$4.00 © 1994 IEEE

start codon in a prokaryotic organism. But recognizing the translation initiation site is still not straightforward: The location of Shine-Delgardo sequences can vary relative to the start codons, and the nucleotides in the region vary as well.

An early application of machine learning to molecular biology involved training *perceptrons* to recognize translation initiation sites in the DNA of the bacterium *E. coli*.[1] (Although translation initiation sites are features of mRNA, we can easily recognize them in DNA sequences by applying our knowledge of how a given DNA sequence is transcribed to an RNA sequence.) A perceptron is a neural network with only one output unit and no hidden units. The input units represent the problem's features and their possible values. For example, the input units in Figure 2 represent three features (the nucleotides in the window) with four possible values each (A, C, G, or T).

The state of each unit is called its *activation*, and is typically a real-valued number in the range [0,1]. In Figure 2, the shaded input units have activations of 1, indicating that those feature instances have those values; the others have activations of 0. Real-valued weights connect the input units to the output unit. The output unit's activation for a given feature instance p is computed with an *activation function*:

$$a_{pi} = \begin{cases} 1 \text{ if } \sum_j w_{ij} a_{pj} > \theta \\ 0 \text{ otherwise} \end{cases}$$

where a_{pi} is the activation of the ith unit in response to instance p, w_{ij} is the weight connecting unit j to the output unit i, and θ is a threshold. The output activation is then considered the perceptron's "answer." For example, an activation of 1 typically indicates a positive instance (such as a translation initiation site), whereas an activation of 0 indicates a negative instance.

Teaching a perceptron involves adjusting the network weights and threshold to maximize the number of training instances that it correctly classifies. Specifically, it makes several passes through the training set, and its weights are updated for each instance:

$$\Delta w_{ij} = \eta(t_{pi} - a_{pi})a_{pj}$$

where t_{pi} is the *teaching signal* (correct response) for instance p, and η is a step-size parameter that determines the learning rate.

One way to think about a perceptron is as a matrix whose rows represent A, C, G, and T, and whose columns represent positions in the window. Each element of this matrix is a number that represents the associated weight for a particular nucleotide in a particular window position. Descriptions of such weight matrices are common in the biological literature. Another way to think about a perceptron is as an $(n-1)$-dimensional hyperplane where n is the number of input units. An activation pattern on the input units corresponds to a point in the n-dimensional space, and the class that the perceptron predicts is determined by which side of the hyperplane the point is on. Thus, a perceptron can accurately represent only concepts that are *linearly separable*; that is, concepts for which a hyperplane can completely separate the positive and negative instances.

In training perceptrons to recognize translation initiation sites, Stormo and his colleagues experimented with windows that were 101, 71, and 51 nucleotides wide.[1] As in Figure 2, they used four input units to represent each nucleotide in the window. The set of positive instances was 124 known initiation sites, and the set of negative instances was 167 sites that a rule-based technique had falsely identified as initiation sites. The researchers aligned the positive instances so that the start codon of each instance occupied the same window positions. They found that the perceptron with the 101-nucleotide window generalized best. Not surprisingly, the most significant weights were those connected to the units representing the initiation codon and the nucleotides in the Shine-Dalgarno region.

Since this early work, researchers have developed backpropagation and other learning algorithms for *multilayer* networks (with hidden units) and have applied these networks to recognize translation initiation sites and other signals. Hidden units can transform the space defined by input unit activations into one in which output units can more profitably make linear discriminations, thereby enabling more complex concept descriptions than perceptrons allow.

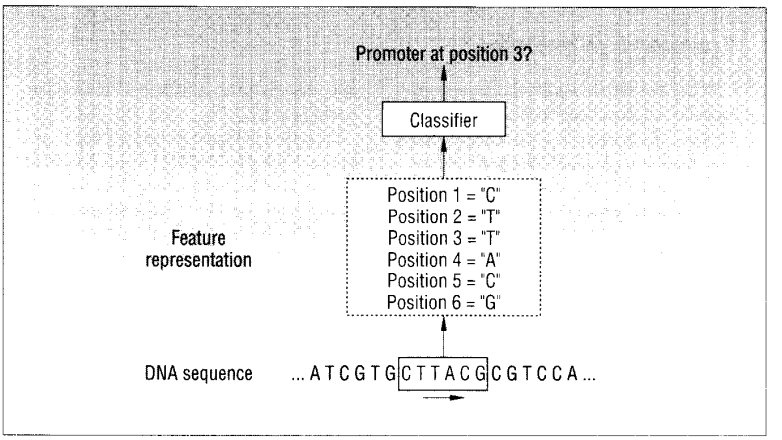

Figure 1. Search by signal as classification.

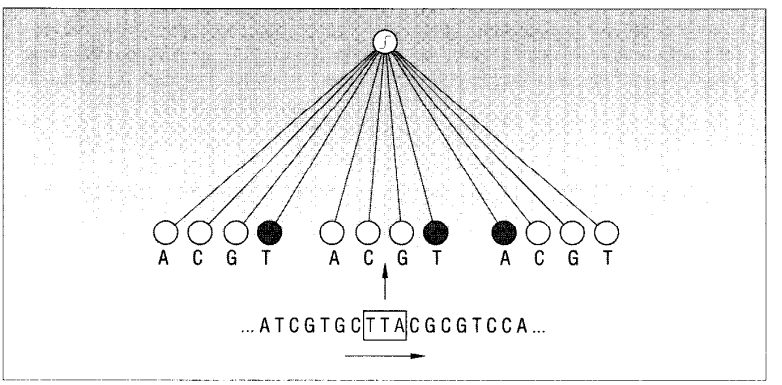

Figure 2. A perceptron.

From DNA to protein

A DNA molecule usually comprises two strands that coil around each other into a double helix. Each strand is a linear sequence composed from four different *nucleotides*—adenine, guanine, thymine, and cytosine—commonly abbreviated as A, G, T, and C. The two strands are held together by bonds that connect each nucleotide to its *complementary* nucleotide on the other strand: A always bonds to T, and C always bonds to G.

Certain subsequences of a DNA strand, called *genes*, are blueprints for *proteins*, which provide most of a cell's structure, function, and regulatory mechanisms. Proteins are also linear sequences; they are composed from among 20 *amino acids*. Between the genes are *noncoding regions* that do not encode proteins.

The process by which genes produce proteins is called *gene expression*. The process is somewhat different for *prokaryotic* organisms, such as bacteria, which lack cell nuclei, and higher, or *eukaryotic*, organisms. Here we discuss only the differences that are germane to finding genes.

As shown in Figure A, the first step in gene expression is *transcription*, which uses DNA as a template to synthesize an RNA molecule. RNA is similar to DNA: Each ribonucleotide of an RNA strand matches the DNA from which it was transcribed, except that a uracil (U) nucleotide replaces each thymine nucleotide. The synthesized RNA is therefore complementary to one strand of the DNA and identical to the other strand (except for T → U substitutions). We follow biological convention here by referring to the gene as being on the strand that is identical to the RNA.

The transcription from DNA to RNA is performed by the enzyme *RNA polymerase*. It begins transcription after it binds to a *promoter*, a regulatory *signal* on a DNA molecule. (A signal is a localized region of DNA that has a specific function.) In eukaryotic DNA, each gene is transcribed independently, so there is a promoter before every gene. In prokaryotic DNA, several consecutive genes may be transcribed into a single, continuous RNA molecule, so a promoter does not necessarily precede each gene.

The *translation* process uses the RNA strand as a template to synthesize a protein molecule. RNA used this way is called *messenger RNA* (mRNA). A complex molecule called a *ribosome* "reads" an mRNA strand and uses each string of three consecutive nucleotides in mRNA to encode a single amino acid. These triplets are *codons*, and the mapping from codons to amino acids is the *genetic code*. The nucleotides can be grouped into triplets in three ways: a given nucleotide can occupy the first, second, or third position in a codon. The ribosome reads only one of the groupings, which is the gene's *reading frame*. (As an analogy, a bit stream that contains a message encoded in ASCII has eight possible reading frames, and the correct frame must be known to decode the message.)

There are three special codons, called *stop codons*, that cause the translation process to terminate. Unlike the other codons, which are translated to amino acids, stop codons signal the ribosome to release the mRNA chain, thus terminating translation.

In eukaryotic organisms, certain nucleotide sequences are spliced out of the mRNA before it is translated to protein (see Figure B). Thus, genes in eukaryotes consist of alternating segments of *exons*, the sequences that are expressed, and *introns*, the sequences that are spliced out. Introns range in length from fewer than 100 to more than 1,000 nucleotides. The boundary points where splicing occurs are *splice junctions*.

An organism's *genome* is the complete complement of DNA found in each of its cells. The human genome contains about 6 billion nucleotides and 100,000 genes. The genome is often called an organism's blueprint because each gene is a plan for a protein, and proteins are an organism's key building blocks. However, unlike a blueprint, a large part of the genome does not contain such plans, but contains sequences that regulate protein construction, and sequences that may have no useful function. So, a fundamental problem in analyzing DNA sequences is locating those plans.

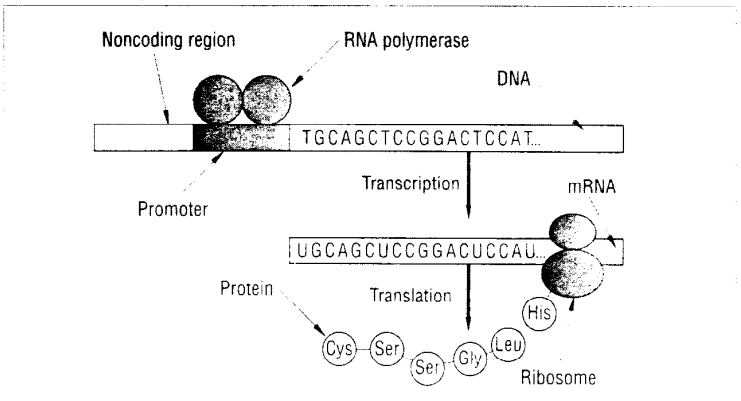

Figure A. Gene expression.

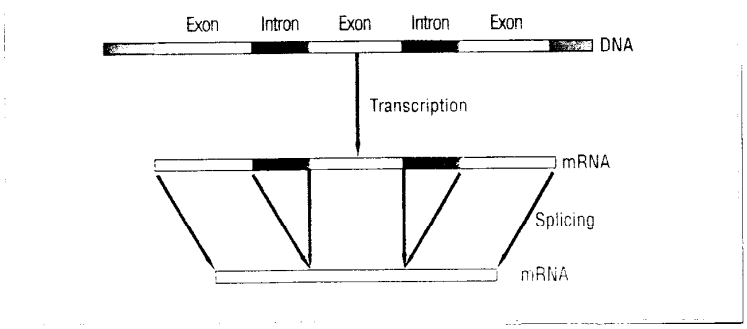

Figure B. mRNA splicing in eukaryotic organisms.

Further reading

1. N.G. Cooper, ed., *Los Alamos Science, Number 20: The Human Genome Project*, Los Alamos Nat'l Laboratory, Los Alamos, N.M., 1992.
2. J.D. Watson et al., *Molecular Biology of the Gene*, Vol. 1, Benjamin Cummings, Menlo Park, Calif., 1987.

```
promoter :-     contact, conformation.
contact :-      minus_35, minus_10.

minus_35 :-     @-37 "cttgac".
minus_35 :-     @-37 "xttgxca".
minus_35 :-     @-37 "xttgaca.
minus_35 :-     @-37 "xttgac".

minus_10 :-     @-14 "tataat".
minus_10 :-     @-14 "xtaxaxt".
minus_10 :-     @-14 "xtataat".
minus_10 :-     @-14 "xxtaxxxt".

conformation :- @-45 "aaxxa".
conformation :- @-45 "axxxxa", @-4 "t",
                @-28 "txxxtxaaxxtx".
conformation :- @-49 "axxxxt", @-1 "a",
                @-27 "txxxxaxxtxtg".
conformation :- @-47 "caaxttxac", @-22 "gxxxtxc",
                @-8 "gcgccxcc".
```

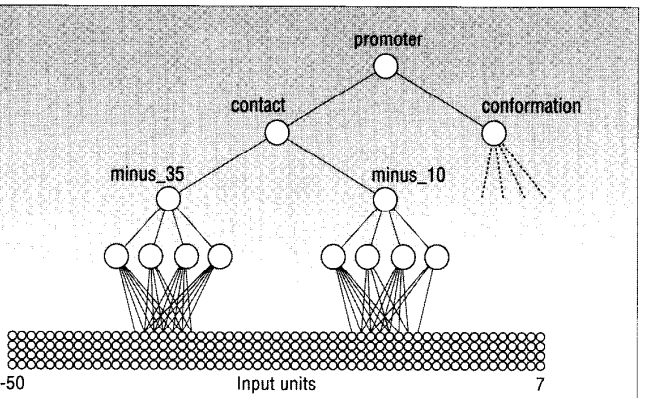

Figure 3. The promoter rules and initial neural network.

Transcription initiation sites. Promoters are another type of signal that is useful for locating genes in DNA because transcription begins just downstream from where RNA polymerase binds to a promoter. Several research groups have investigated using neural networks to recognize promoters.

Towell, Shavlik, and Noordewier have used the *Kbann* algorithm, a novel approach that combines neural network and symbolic learning.[2] The algorithm initializes a neural network's topology and weights using a set of approximately correct, propositional rules. Ordinary neural network learning techniques then adjust the weights. Compared with conventional networks — where the weights are initially assigned small random values, and a suitable topology is determined through experimentation — Kbann provides a way to use problem-specific knowledge during learning. The resulting networks often learn faster and, more importantly, find solutions that result in better generalization.

The first real-world problem to which Towell and Shavlik applied their algorithm was recognizing promoters in *E. coli* DNA. They used a window of 57 nucleotides, and they aligned the positive instances (promoter sequences) so that each instance's transcription initiation site occurred seven nucleotides from the window's right edge. Noordewier, a computational biologist, derived an approximately correct rule set for recognizing *E. coli* promoters from the biological literature. These rules identified two sets of sequence patterns that should occur about 10 and 35 nucleotides upstream from where transcription begins. These two pattern sets, commonly referred to as the *-10* and *-35* regions, are where RNA polymerase binds to the DNA sequence and are widely accepted as defining characteristics of promoters. (The rule set also specified patterns for several other upstream regions of controversial significance: These *conformation* rules try to capture the effect of DNA's helical structure on the spatial alignment of the *-10* and *-35* regions.)

Figure 3 shows the promoter rule set (in a Prolog-like syntax) and an initial Kbann network (links with small weights are not shown). The notation @-37 "cttgac" indicates that the rule is looking for the sequence C-T-T-G-A-C, starting 37 nucleotides before the putative transcription initiation site. An "x" indicates that any nucleotide at that position will match the rule.

Although this promoter rule set represented textbook characteristics of promoters, it did not correctly classify any of the promoter sequences in the set of instances used to train and test the algorithm. Neural-network training, however, refined the rules so that they more accurately represented the essential characteristics of the promoters.

The researchers found that networks initialized by the Kbann algorithm generalized better than conventional neural networks, decision trees, and nearest-neighbor classifiers (we'll discuss these later). Their results indicate that Kbann's approximately correct, task-specific rules assist learning by identifying important problem features and their significant relationships. They also discovered that the networks learned to discard the conformation rules during training, indicating that those rules do not represent a salient aspect of promoters.

Splice junctions. Because eukaryotic genes may contain introns, determining the extent of coding regions in their DNA involves more than just finding start and stop codons. The splice junctions must also be located and classified. Identifying these junctions is important because, to determine the protein that a gene produces, it is necessary to precisely demarcate the segments of the DNA sequence that are eventually translated.

Lapedes and his colleagues used several approaches — neural networks, decision trees, and *k*-nearest neighbor classifiers — to recognize splice junctions in human DNA.[3] They used the ID3 algorithm[4] to induce decision trees (see Figure 4). Each internal node in a decision tree represents a test applied to one of the problem features. The branches emanating from a node represent the test's possible outcomes. With nominal features, the test commonly results in a branch for each possible feature value. Each leaf represents a predicted class. In Figure 4 the classes are *donor* (an exon/intron border) and *negative* (not a donor).

Classification using a decision tree involves following a path from the root down to a leaf, using the decisions made at each node to determine which branches to follow. Decision-tree learning is a recursive process that involves adding nodes until the tree sufficiently separates the training data by class. The ID3 learning algorithm uses an information-theoretic measure to determine which feature to branch on at each node, and then makes recursive calls to build subtrees for each created branch. It uses all the training instances to select the test at the root node, but it uses smaller subsets of the training data to select the tests at subsequent nodes. Specifically, as ID3 constructs the tree, it also uses the tree to classify the training instances; it uses only those training instances that reach a node to select the test at that node. An ad-

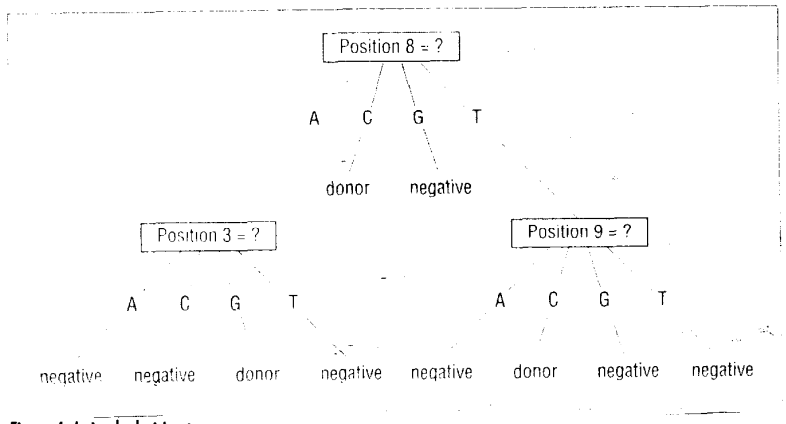

Figure 4. A simple decision tree.

vantage of using decision trees is that they can be transformed easily into sets of rules.

Lapedes and his colleagues also investigated a *k*-nearest neighbor approach,[5] a simple learning technique that does not require any training per se. The concept representation is the entire training set, and a new instance is classified by identifying the *k* "nearest" training instances; the class label associated with the majority of these instances is the predicted class. The effectiveness of this approach depends on the metric used to measure the distance between two instances. The researchers cleverly used a weighted Hamming distance. An ordinary Hamming metric defines the distance as the number of window positions in which the instances have different nucleotides. A weighted Hamming distance also associates a weight with each window position. The weight is calculated by an information-theoretic metric that uses the training instances to measure the average amount of information contributed by each window position.

The researchers used windows of 11, 21, and 41 nucleotides to evaluate the neural network, decision tree, and nearest-neighbor approaches. They trained separate classifiers to recognize exon/intron borders (*donors*) and intron/exon borders (*acceptors*). The instances were aligned so the splice junctions were in the center of the window. Donors and acceptors have a pair of nucleotides that is *highly conserved* (G-T for donors and A-G for acceptors) on the intron side of the splice junction. A highly conserved sequence is one that occurs with high frequency in a given location. The negative training instances, which the researchers took from known exons, were selected so that they had A-G or G-T in the center of the window. This prevented the classifiers from learning a trivial distinction such as "A-G in the center of the window indicates acceptor."

Lapedes and his colleagues found that neural networks generalized better than decision trees or *k*-nearest neighbor classifiers. The acceptor-recognition networks correctly classified 91 percent of the test set instances, and the donor-recognition networks correctly classified 95 percent. Although the decision trees were not as accurate as the neural networks, their concept representations were more comprehensible. The splice-junction trees were transformed into rule sets that were found to be relatively small and biologically interpretable.

Search by content

Like the search-by-signal methods we've described, many search-by-content methods slide a fixed-sized input window along a sequence to generate predictions for the entire sequence. But unlike search by signal, which looks for specific functional sites in DNA, search by content identifies genes by recognizing general patterns in their nucleotide sequences. For prokaryotic DNA, this involves distinguishing genes from the noncoding regions that are interspersed between them. For eukaryotic DNA, the goal is not only to distinguish genes from intergenic noncoding regions, but also to distinguish introns from exons. Search-by-content methods address three questions: Which regions are coding and, for a given region, which strand and which reading frame encode the protein? (As noted in the sidebar, a gene's *reading frame* refers to how consecutive nucleotides are grouped into triplets.)

Search by content takes advantage of several properties that can significantly distinguish coding regions from introns and noncoding regions:

- By definition, a coding region encodes a protein, so the fact that some amino acids appear in proteins more frequently than others influences the nucleotide composition of coding regions.
- A protein's shape largely determines its function, and that shape is partly determined by electrostatic interactions among neighboring amino acids. So, some amino acids are more likely to be neighbors than others, and thus some codons are more likely to be neighbors.
- Due to the *degeneracy* of genetic code, there are different numbers of codons for different amino acids. There are 64 different codons, since there are four different nucleotides and each codon consists of three nucleotides. Sixty-one of the codons map to amino acids; the other three are the stop codons. There are, however, only 20 amino acids. Consequently, many amino acids are encoded by several different codons.
- The codons that map to an amino acid are not used equally in most organisms. This bias is the organism's *codon preference*.
- Coding regions cannot contain stop codons.

Bayesian approaches. Several search-by-content methods, such as Staden and McLachlan's *codon usage method*,[6] are based on Bayes' theorem. Given a window of nucleotides, their approach estimates the probability that each of a strand's three reading frames encodes a protein. For a sequence S in the window, the probability that frame i is coding (C_i) is

$$P(C_i|S) = \frac{P(S|C_i) \times P(C_i)}{\sum_{f=1}^{3} P(S|C_f) \times P(C_f)}$$

The prior probability that each frame is coding, $P(C_i)$, is estimated as the number of triplets in the window in frame i, divided by the number of triplets that can be formed in the window in all three frames. As the window size increases, $P(C_i)$ approaches 1/3 for all three frames. Each conditional probability, $P(S|C_i)$, is the probability that we would get sequence S if we arbitrarily selected a coding sequence the same length as S.

These conditional probabilities are estimated by compiling a table of the frequencies of each codon in the organism's known genes. Each codon's frequency value is an estimate of the conditional probability that the codon occupies a given position in se-

quence S, given that S encodes a protein. Staden and McLachlan make the simplifying assumption that a gene's codons are independent of each other, and thus arrive at the estimate

$$P(S|C_i) = \prod_{j=1}^{n} P(S_i(j)|C_i)$$

where $S_i(j)$ is the jth triplet in frame i in sequence S, and n is the number of triplets in frame i in S. That is, to determine the probability of finding sequence S in a coding region, we calculate the joint probability of finding S's individual codons in such a region.

This approach assumes that the given sequence encodes a protein in one of the three reading frames on the strand under consideration. Although this assumption is not generally valid because the window may be positioned over a noncoding region, the approach still works well in practice. Moreover, it is straightforward to extend the codon usage method to consider *noncoding* as a hypothesis. This requires estimating the prior probability of the *noncoding* hypothesis and the conditional probabilities of each codon, given the *noncoding* hypothesis. Estimating these is problematic for some species, however. Sequencing efforts often concentrate on areas that are dense with genes, so there may be a dearth of noncoding sequence data. Also, it is sometimes difficult to ascertain that a stretch of putative noncoding DNA does not actually contain a gene.

Typically, researchers use the codon usage method to generate a plot for each reading frame (see Figure 5). Each plot is a series of connected points that represents the predicted probability that a frame encodes a protein. Sharp changes in these plots indicate coding-region boundaries. For example, in Figure 5, the start of the first frame's coding region corresponds to a steep increase in the topmost plot's predicted probabilities. Although this method assumes that a given sequence encodes a protein in one of its reading frames, it can usually help detect noncoding regions because they tend to produce wildly fluctuating predictions, whereas coding regions produce consistently high probabilities.

These plots can also detect *frameshift errors*: laboratory errors during sequencing that insert or delete nucleotides in the sequence data. Because of the genetic code's triplet nature, a frameshift error can have a devastating effect on the prediction of the amino acid sequence translated from a gene. Once the computed translation is out of frame, the predicted protein will bear no resemblance to the actual protein. However, we can often detect frameshift errors by noting a sharp drop in the plot of one reading frame accompanied by a steep increase in another. Figure 5 shows a frameshift error that shifts the predicted reading frame from frame 1 to frame 2.

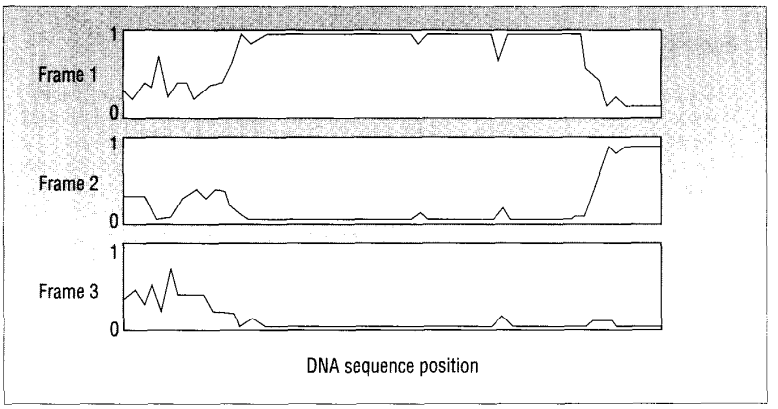

Figure 5. Reading frame plots.

Borodovsky and McIninch have applied a related statistical method, Markov chain models, to gene recognition.[7] Like the Bayesian method, this approach computes a given sequence's likelihood in each reading frame and in noncoding DNA. In this approach, however, a DNA sequence can be thought of as being generated by a state-based model. Borodovsky and McIninch use a four-state model where each state corresponds to one of the four nucleotides. They calculate the prior probabilities of the states and the probabilities of the transitions from the sequences in training set. They then calculate a sequence's likelihood as the product of the initial state probability (the probability of the sequence's first element) and the probabilities of successive state transitions. Markov chain models can use statistics that describe *sequences* of transitions through several states: A kth-order model uses statistics that describe transition chains that link $k+1$ states.

Neural network approaches. Perhaps the most problematic assumption of the codon usage method is that the codons in the window are independent. Neighboring codons are certainly *not* independent: Interactions among neighboring amino acids partly determine a protein's shape, and hence its function.

Farber, Lapedes, and Sirotkin showed that accounting for the joint probabilities of neighboring codons can produce better coding-region predictions.[8] In one experiment, they compared the prediction accuracy of a Bayesian method with that of perceptrons using windows that ranged from 5 to 90 codons long. The Bayesian method formulated the prediction task as a two-class problem: Given a sequence, it determined whether it occurs in an intron or an exon. This assumes that the gene's reading frame is known; the classifiers must simply distinguish introns from exons. Given a sequence S, the probability that S is in an exon is

$$P(E|S) = \frac{P(S|E) \times P(E)}{P(S|E) \times P(E) + P(S|I) \times P(I)}$$

where E represents exon, and I represents intron. The conditional probabilities $P(S \mid E)$ and $P(S \mid I)$ are estimated using the independence assumption and codon frequencies tabulated from sets of known introns and exons.

The perceptrons used the same feature representation as the Bayesian approach: Sixty-four features represent each codon's frequency of occurrence. These features are represented using 64 input units, so that the activation of each input unit is effectively a count of the number of times that the corresponding codon occurs in the window.

The researchers found that the perceptrons were significantly more accurate, especially with larger windows, because the assumption of codon independence does not bind them. They showed that a perceptron's weights can be set by hand so that it calculates the same probabilities (their networks had continuous activation functions) as Bayes' theorem under the independence assumption. Although these weights are not optimal when the independence assumption is not true (as in this problem), the perceptron-training algorithm can find optimal weights for the given training instances and feature representation, even when the assumption is violated.

Empirical learning

Empirical learning (also called *supervised learning*, *learning from examples*, and *similarity-based learning*) is an inductive process that forms a general description of a *target concept*, using a set of known *positive* instances of the concept and, usually, a set of *negative* instances known to not belong to the concept class. (Some tasks involve more than two classes. In such a case, each instance is labeled by its class.) These sets compose a *training set*. Inductive learning aims to synthesize a concept description that can correctly classify the training instances and (most importantly) novel instances that are not in the training set. The ability to classify previously unseen instances is called *generalization*.

For example, we might want to learn the concept of poisonous mushrooms. The positive examples are known poisonous species, and the negative examples are known edible species. We are most concerned that our classifier correctly identifies newly found species as poisonous or edible.

Empirical-learning methods are characterized by an instance-representation language, a concept-representation language, a learning algorithm, and a classification algorithm. The *instance-representation language* is used to describe the instances processed during training and classification. Often, a fixed-length list of feature-value pairs represents instances. For example, the following feature-value pairs describe one mushroom instance:

```
[cap-shape = conical,
 odor = almond,
 gill-attachment = free]
```

cap-shape, odor, and gill-attachment are the features; conical, almond, and free are the corresponding values.

Using such a language, we must select the features that are potentially relevant to learning the target concept, and specify each feature's type. Real-world instances of the problem are then mapped to this "feature space" so the learning algorithm can process them. Common feature types include Boolean, real, and nominal. (A nominal feature is one whose possible values are not ordered. In our example, gill-attachment is a nominal feature whose possible values are attached, descending, free, or notched.) For training, the instance representation also specifies each instance's class.

The *concept-representation language* defines the space of possible concepts that can be represented by the learning algorithm. The language's richness determines the range of concepts it can represent. For example, the language of first-order logic has more expressive power than that of propositional conjunctions. The richness of the concept-description language determines the number of concept descriptions that are likely to provide a good *fit* to the training data, as well as the complexity of searching the concept description space.

"Fit" is the degree to which the concept description correctly classifies the training instances. When there are many concept descriptions that fit the training set, there is a high probability that the learning algorithm will find a description that does not generalize well. Poor generalization (*overfitting*) results when the concept description captures too much information about the specific training instances and not enough about the concept's general characteristics.

The *learning algorithm* searches the concept representation space to find a description that covers most or all positive instances and few or no negative instances. For many real-world problems, it is not possible to cover all positive and no negative instances because the concept representation language is not rich enough or there is noise in the training data. Noise may result from error or imprecision in measuring feature values or assigning class labels to instances. Noise may also occur when the mapping of real-world objects to instances in the instance-description language is many-to-one. Even if it is possible to find a concept description that fits all of the training instances, it is not necessarily desirable. To avoid overfitting, a simple description that does not fit all the training examples is often preferable to a complex one that does.

The *classification algorithm* takes two inputs: a learned concept description, and instances described using the instance-representation language. The algorithm outputs a prediction of an instance's class or a probability distribution that indicates how likely it is that an instance is a member of each class.

To evaluate how well a classifier has learned a target concept, it is important to measure how it generalizes to instances that it has never seen. This is typically estimated by setting aside a *test set* of instances before training. Unlike training instances, the test set is used not for learning the concept description, but instead to get an unbiased estimate of the trained classifier's prediction accuracy. More sophisticated methods such as cross-validation are sometimes used to better estimate how well an algorithm generalizes.

Further reading

1. J.W. Shavlik and T.G. Dietterich, *Readings in Machine Learning*, Morgan Kaufmann, San Mateo, Calif., 1990.
2. S.M. Weiss and C.A. Kulikowski, *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*, Morgan Kaufmann, San Mateo, Calif., 1991.

In a second experiment, Farber and his colleagues trained perceptrons that captured some codon dependencies in their feature representation. The features were all of the possible *dicodons*; that is, adjacent pairs of codons. There are 64 codons, so there are 4,096 dicodons; the perceptrons had 4,096 corresponding input units. The researchers used the same training sets and window sizes as in the first experiment, but found that these perceptrons generalized significantly better than those that used only codon features. This illustrates a common theme in machine learning research: A learning system's ability to find a good solution to a problem depends highly on the representation used for the problem's features. Even when the researchers added hidden units to the networks that used the single-codon feature representation, those networks did not learn to represent dicodon frequencies as well as the networks that used the dicodon feature representation. Similarly, using a feature representation of the individual nucleotides in the input window resulted in networks that did not generalize as well as those that used a codon feature representation.

Uberbacher and Mural have also applied neural networks to recognizing coding regions in eukaryotic DNA.[9] Their *coding recognition module* (CRM) is a component of Grail, an automated sequence-analysis server.[10] The heavily used server accepts DNA sequences via electronic mail, analyzes them, and then returns e-mail messages describing its results. (A document available by anonymous ftp provides addresses and brief descriptions of Grail and other e-mail servers that perform DNA analysis. The file is pub/databases/info/ serv_ema.txt, at expasy.hcuge.ch.)

Uberbacher and Mural's research has also focused on finding features that lead to good coding region predictions. Seven algorithms called *sensors* calculate the input features of the coding recognition module by evaluating seven aspects of a DNA sequence, including the frequency with which each nucleotide occupies each position in a codon, the likelihood of finding the window's dicodons in coding and noncoding DNA, and the simi-

larity of the sequence to repetitive patterns in noncoding regions. Each sensor indicates the sequence's coding potential. During training, the coding recognition module learns to weight the individual sensors and recognize meaningful correlations in their values. Uberbacher and Mural evaluated their coding recognition module using 19 human genes that were not in the training set; it located 90 percent (71 of 79) of the genes' long (more than 100 nucleotides) exons.

Grail also uses modules that predict splice junctions and translation initiation sites. An expert system with a blackboard control structure assembles the predictions of the individual modules into coherent predictions of the location and intron/exon structure of genes.

Case-based approaches. Researchers have also applied the indexing and retrieval aspects of case-based reasoning to gene recognition: These systems take a new nucleotide or amino acid sequence (a *query sequence*) and search the case memory for similar sequences. They interpret a significant partial match between the query sequence and an element of the case memory as a prediction of a coding region in the query sequence. The case memory is usually not limited to sequences from the same organism as the query sequence because, due to evolution, many different species have highly similar genes. A matching gene in the case memory can also provide insight into the function of a newly discovered gene, which distinguishes this approach from the others we've discussed.

The effectiveness of a case-based algorithm hinges on its method for assessing sequence similarity. The two most important aspects of similarity determination are the sequence level at which comparisons are made, and the algorithm used to measure similarity. The level of comparisons refers to whether the algorithm compares untranslated nucleotide or translated protein (amino acid) sequences. Sequence comparisons are more commonly made at the protein level because that level determines a gene's function. Differences at the nucleotide level do not necessarily indicate differences at the protein level; due to the genetic code's degeneracy, different nucleotide sequences can map to the same amino acid sequence. There are biologically justified scoring schemes, based on evolutionary and chemical similarity, for measuring the similarity of pairs of amino acids.

Unlike the other gene-recognition approaches we discuss, the case-based approach does not use a fixed-size list of features to describe instances. Instead, it compares a query sequence of arbitrary size to case sequences of varying sizes. Although dynamic programming methods can find the optimal partial match between two different-sized sequences, these methods are too expensive for large-sequence databases. There are, however, several fast approximations to dynamic programming that are commonly used to search for similar sequences.[11]

So far, we have assumed that each case is an entire protein sequence. Another approach is to store protein *domains* as cases. Domains are amino acid sequences that act as modular components of proteins. Domains are analogous to subroutines in programs: Each has a specific function, and different combinations of domains (subroutines) give rise to different proteins (programs). This approach provides finer-grained units for predicting a query sequence's function.

How can a case memory of proteins and domains be assembled from a database of protein sequences? Hunter, Harris, and States developed an *unsupervised* learning system that clusters related amino acid sequences into domains and "families" of proteins.[12] Unlike the *supervised* learning methods on which we have focused so far, unsupervised learning approaches are not told what the "correct" classes are; they form their own class definitions. Unsupervised learning aims to cluster the training set so that similar instances are in the same class, and dissimilar instances are in different classes. In an experiment of impressive scale, Hunter, Harris, and States applied their method to a set of more than 60,000 protein sequences. Their unsupervised algorithm formed about 12,000 clusters; some of these corresponded to protein families, some represented functional domains, and some contained a mixture of whole and partial proteins. They have also developed ClassX, a tool for matching novel sequences against a case memory consisting of the clusters formed by their algorithm.[13]

Combined methods

Although we have discussed the search-by-signal and search-by-content methods separately, the most promising approaches combine predictions of several different signals and coding regions. Grail (described previously) is one such approach; another is the GeneId system, which predicts start codons, stop codons, donor sites, and acceptor sites, and then assembles these predictions into possible genes.[14] GeneId, like Grail, is publicly available as an e-mail server on the Internet.

Snyder and Stormo's GeneParser system also integrates signal and content predictions to identify introns and exons.[15] A dynamic-programming algorithm predicts the extent of individual exons and introns in a given DNA sequence. This method uses two arrays that contain estimates of the likelihood that each subsequence of a given sequence is an intron or an exon. Neural networks, which take as input both content and signal measures, calculate those estimates.

ALTHOUGH MACHINE LEARNING approaches have shown great promise at recognizing genes in uncharacterized DNA, there is still room for improvement. It is difficult to train signal classifiers to be both highly sensitive (predicting few false negatives) and highly specific (predicting few false positives). Similarly, search-by-content methods often fail to recognize short exons.

However, applying machine learning to gene recognition is fruitful for both molecular biologists and computer scientists. Biologists gain effective, automated methods for analyzing sequence data; machine learning researchers gain important, real-world testbeds. (Data sets for several of the gene-finding problems we've discussed are available by anonymous ftp from the University of California at Irvine's Repository of Machine Learning Databases and Domain Theories: ftp.ics.uci.edu.) Because of these mutual interests, *computational biology* is a rapidly growing field.[16] We expect that this marriage of computer science and biology will continue to advance the state of the art in both fields.

Acknowledgments

Our research is partially supported by Department of Energy Grant DE-FG02-91ER61129, National Science Foundation Grant IRI-9002413, and Office of Naval Research Grant N00014-93-1-0998. We thank Carolyn Allex, Rich Maclin, and Steve Gallant for their helpful comments. This article was submitted to *IEEE Expert* on March 8, 1993, and was accepted for publication on December 1, 1993.

References

1. G.D. Stormo et al., "Use of the Perceptron Algorithm to Distinguish Translational Initiation Sites in *E. coli*," *Nucleic Acids Research*, Vol. 10, No. 9, 1982, pp. 2997–3011.
2. G. Towell, J. Shavlik, and M. Noordewier, "Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks," *Proc. Eighth Nat'l Conf. Artificial Intelligence*, AAAI Press, Menlo Park, Calif., 1990, pp. 861–866.
3. A. Lapedes et al., "Application of Neural Networks and Other Machine Learning Algorithms to DNA Sequence Analysis," in *Computers and DNA*, SFI Studies in the Sciences of Complexity, Vol. VII, G. Bell and T. Marr, eds., Addison-Wesley, Reading, Mass., 1989, pp. 157–182.
4. J.R. Quinlan, "Induction of Decision Trees," *Machine Learning*, Vol. 1, 1986, pp. 81–106
5. T.M. Cover and P.E. Hart, "Nearest-Neighbor Pattern Classification," *IEEE Trans. Information Theory*, Vol. 13, No. 1, 1967, pp. 21–27.
6. R. Staden and A.D. McLachlan, "Codon Preference and Its Use in Identifying Protein Coding Regions in Long DNA Sequences," *Nucleic Acids Research*, Vol. 10, No. 1, 1982, pp. 141–156.
7. M. Borodovsky and J. Mcininch, "Prediction of Gene Locations Using DNA Markov Chain Models," *Proc. Second Int'l Conf. Bioinformatics, Supercomputing, and Complex Genome Analysis,* World Scientific, Singapore, 1993, pp. 231–248.
8. R. Farber, A. Lapedes, and K. Sirotkin, "Determination of Eucaryotic Protein Coding Regions Using Neural Networks and Information Theory," *J. Molecular Biology*, Vol. 226, No. 2, 1992, pp. 471–479.
9. E.C. Uberbacher and R.J. Mural, "Locating Protein Coding Regions in Human DNA Sequences by a Multiple-Sensor—Neural Network Approach," *Proc. Nat'l Academy of Sciences*, Vol. 88, No. 24, 1991, pp. 11261–11265.
10. E.C. Uberbacher et al., "Gene Recognition and Assembly in the Grail System: Progress and Challenges," *Proc. Second Int'l Conf. Bioinformatics, Supercomputing, and Complex Genome Analysis,* World Scientific, Singapore, 1993, pp. 465–476.
11. S.F. Altschul et al., "Basic Local Alignment Search Tool," *J. Molecular Biology*, Vol. 215, No. 3, 1990, pp. 403–410.
12. L. Hunter, N. Harris, and D.J. States, "Efficient Classification of Massive, Unsegmented Datastreams," *Proc. Ninth Int'l Conf. Machine Learning*, Morgan Kaufmann, San Mateo, Calif., 1992, pp. 224–232.
13. N.L. Harris, D.J. States, and L. Hunter, "ClassX: A Browsing Tool for Protein Sequence Megaclassification," *Proc. 26th Hawaii Int'l Conf. System Sciences*, IEEE Computer Society Press, Los Alamitos, Calif., 1993, pp. 554–563.
14. R. Guigo et al., "Prediction of Gene Structure," *J. Molecular Biology*, Vol. 226, No. 1, 1992, pp. 141–157.
15. E.E. Snyder and G.D. Stormo, "Identification of Coding Regions in Genomic DNA Sequences: An Application of Dynamic Programming and Neural Networks," *Nucleic Acids Research*, Vol. 21, No. 3, 1993, pp. 607–613.
16. L. Hunter, D. Searls, and J. Shavlik, eds., *Proc. First Int'l Conf. Intelligent Systems for Molecular Biology*, AAAI Press, Menlo Park, Calif., 1993.

Mark W. Craven is a PhD student and research assistant in the Department of Computer Sciences at the University of Wisconsin—Madison. His research interests center around machine learning and computational biology. He holds an MS in computer science from the University of Wisconsin and a BA in political science from the University of Colorado. He is a member of AAAI and ACM. He can be contacted at the Dept. of Computer Sciences, Univ. of Wisconsin-Madison, 1210 West Dayton Street, Madison, WI 53706; Internet craven@cs.wisc.edu

Jude W. Shavlik is an associate professor in the Department of Computer Sciences at the University of Wisconsin—Madison. His research interests include machine learning, neural networks, and computational biology. He received his PhD in computer science at the University of Illinois in 1987, his MS in molecular biophysics and biochemistry from Yale in 1980, and his BS in electrical engineering and BS in life sciences from MIT in 1979. He is a member of IEEE, ACM, and AAAI. He can be contacted at the Dept. of Computer Sciences, Univ. of Wisconsin-Madison, 1210 West Dayton Street, Madison, WI 53706; Internet shavlik@cs.wisc.edu