Revised version submitted (3/94) to *IEEE Expert* (Special Track on AI and Molecular Biology).

Machine Learning Approaches to Gene Recognition

Mark W. Craven
Jude W. Shavlik
Computer Sciences Department
University of Wisconsin
1210 West Dayton St.
Madison, Wisconsin 53706
phone: (608) 263-0475
email: {craven, shavlik}@cs.wisc.edu

Abstract

Currently, a major computational problem in molecular biology is to identify genes in uncharacterized DNA sequences. The variation, complexity, and incompletely-understood nature of genes make it impractical to hand-code algorithms to recognize them. Machine learning methods – which are able to form their own descriptions of genetic concepts – offer a promising approach to this problem. This article surveys machine-learning approaches to identifying genes in DNA. We discuss two broad classes of gene-recognition approaches: *search by signal* and *search by content*. For both classes, we define the specific tasks that they address, describe how these tasks have been framed as machine-learning problems, and survey some of the machine-learning algorithms that have been applied to them.

Keywords: machine learning
computational biology
genetic sequence analysis
neural networks
decision trees
nearest-neighbor methods
Bayesian methods
case-based reasoning

1 Introduction

Biological laboratories around the world are now sequencing large stretches of DNA from humans and other organisms [1]. Determining the nucleotide sequence of a DNA molecule, however, is only a first step toward the ultimate goals of (1) understanding the functionality and (2) knowing the locations of all of the genes and regulatory sites on the molecule. As a result of these sequencing efforts, there is a great volume of sequence data that needs to be analyzed. Because direct laboratory analysis of this data is difficult and expensive, computational techniques are essential to the task of identifying genes in DNA. The variation, complexity, and incompletely-understood nature of genes make it impractical to hand-code algorithms that recognize them. Computational techniques that are able to form their own descriptions of genetic concepts are therefore well suited to DNA sequence analysis. The AI field of *machine learning* involves developing and studying these kinds of automated methods; that is, methods that are able to *learn* a useful description of a target concept (e.g., the concept of a gene) when given instances of the concept, rather than an explicit definition of it. This article provides a survey of machine-learning approaches to recognizing genes in DNA.

The focus of this article is on (1) defining the tasks involved in recognizing genes in nucleotide sequences and (2) describing some of the machine-learning approaches that have been applied to these tasks. We first provide an introduction to a specific paradigm of machine learning, called *empirical learning*, that encompasses the approaches described in this article. We then give a brief introduction to the molecular biology that underlies the problem of finding genes in DNA. The main sections of this article survey machine-learning approaches to identifying genes in DNA. These sections are organized around two primary classes of gene-recognition approaches: *search by signal* and *search by content*. These classes are distinguished by the sequence features on which each approach concentrates. For both classes, we define the tasks that they address, describe how these tasks have been framed as machine-learning problems, and survey some of the machine-learning algorithms that have been applied to them.

2 Empirical Learning

The type of machine learning that we are concerned with in this article is commonly called *empirical learning*. (This type of learning is also sometimes called *supervised learning*, *learning from examples*, and *similarity-based learning*.) Empirical learning is an inductive process that involves forming a general description of a *target concept*, using a set of known instances of the concept, and usually, a set of instances known to *not* belong to the concept class. These sets of *positive* and *negative* instances are collectively referred to as a *training set*. The goal of the inductive learning process is to synthesize a concept description that is able to correctly classify positive and negative instances. For example, we might be interested in learning the concept of *poisonous mushrooms*. In this case, the positive examples are known species of poisonous and the negative examples are known species of edible mushrooms. The concept description should correctly classify the instances of the training set, and more importantly, it should correctly classify novel instances; that is, instances that were not members of the training set. In the mushroom example, clearly we are most interested in having our classifier correctly identify newly-found species of mushrooms as being either poisonous or edible. The ability to classify such previously-unseen instances is referred to as *generalization*. (Often the learning task requires distinguishing instances that belong to more than just two classes. In this case, training instances are not merely labelled *positive* and *negative*, but instead each is labelled with the class to which it belongs.)

Empirical-learning methods are characterized by the following aspects:

- A language (i.e., a notation) for representing instances.
- A language for representing concepts.
- An algorithm for forming a concept description given a set of classified instances.
- An algorithm for using a concept description to classify instances.

We discuss each of these aspects below.

The instances that the learning system processes, both during training or classification, are described using the *instance-representation language*. One type of language that is commonly used to represent instances is a fixed-length list of feature-value pairs. For example, one mushroom instance might be described by the following list of feature-value pairs:

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

In this example, `cap-shape`, `odor`, and `gill-attachment` are the features; `conical`, `almond`, and `free` are the corresponding values. Using such a language, a human must select the problem features that are deemed potentially relevant to learning the target concept, and specify the type of each feature. Real-world instances of the problem are then mapped to this "feature space" so that they can be processed by the learning algorithm. Some common feature types are: Boolean, real, and nominal. (A nominal feature is one for which there is not an ordering over the possible values of the feature. In the mushroom example, `gill-attachment` is a nominal feature with the possible values of `attached`, `descending`, `free`, or `notched`.) Additionally, for training instances, the instance language is used to specify the class to which each instance belongs.

The *concept-representation language* defines the space of possible concepts that can be represented by the learning algorithm. The richness of the concept-description language determines the range of concepts that can be represented by it. For example, the language of first-order logic programs has more expressive power than the language of propositional conjunctions. Naturally, it is desirable that the concept-description language be sufficiently rich that it is able to accurately represent the target concept. The richness of the language also 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 space of concept descriptions. The term *fit* refers to the degree to which the concept description correctly classifies the training instances. A problem that arises when there are many concept descriptions that fit the training set is that there is a high probability that the learning algorithm will find a concept description that does not generalize well. Poor generalization results when the concept description captures too much information about the specific instances in the training set, and not enough information about the general characteristics of the class. This phenomenon is called *overfitting*.

The *learning algorithm* performs a search through the space of concept descriptions in order to find a description that *covers* most (perhaps all) of the positive instances and few (perhaps none) of the negative instances. For many real-world problems, it is not possible to cover all of the positive and none of the negative instances because either the given concept representation language is not rich enough to allow a perfect covering or because there is noise present in the training data. This noise may be attributable to several causes. It may be due to error or imprecision that occurs in measuring feature values or assigning class labels to instances. Alternatively, noise may be introduced in cases where the process of mapping real-world objects to instances in the instance-description language is many-to-one. Even in cases where it is possible to find a concept description that fits all of the training instances, it is not necessarily desirable to do so. In order to avoid overfitting, it is often preferable to choose a simple concept description that does not fit all of the examples over a more complex one that does.

The *classification algorithm* associated with a particular empirical learning method takes two inputs: a learned concept description and instances described using the instance representation language. The classification algorithm produces as output a prediction of the class to which the given instance belongs (or a probability distribution that indicates how likely it is that the instance is a member of each class).

In order to evaluate how well a classifier has learned a target concept, it is important to measure how it generalizes to instances that it has not seen before. This is typically estimated by setting aside a set of instances, called a *test set*, prior to training. Unlike training instances, the instances in the test set are not used in learning the concept description, but instead are used to get an unbiased estimate of the prediction accuracy of the trained classifier. More sophisticated methodologies, such as cross-validation, are sometimes used to gain better estimates of how well an algorithm generalizes.

3 A Brief Introduction to Molecular Biology

This section provides a brief introduction to the molecular biology of the gene. Our description of this topic is quite simplified and ignores many salient aspects of molecular genetics. A thorough treatment of the biology underlying gene recognition can be found elsewhere [1, 2]. Our description emphasizes aspects of the biology that are relevant to finding genes by computational methods.

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

Certain subsequences of a DNA strand, called *genes*, serve as blueprints for *proteins*. Interspersed between the genes are segments, termed *noncoding regions*, that do not encode proteins. Proteins are important because they provide most of the structure, function, and

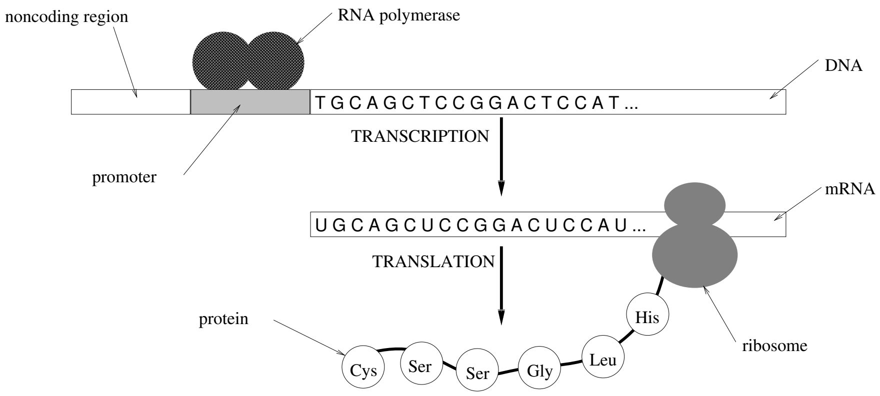

Figure 1: **The process of gene expression.** DNA and RNA are composed of linear chains of nucleotides. *Transcription* involves synthesizing messenger RNA (mRNA) using DNA as a template. The enzyme *RNA polymerase* is the molecule that transcribes DNA into RNA. A site where RNA polymerase binds to DNA to begin transcription is called a *promoter*. Messenger RNA is *translated* to protein by a molecule called a ribosome. Proteins are linear chains of amino acids; each amino acid is encoded by a string of three consecutive nucleotides. *Noncoding regions* are stretches of DNA that do not encode proteins.

---

regulatory mechanisms of cells. Proteins are also linear sequences; they are composed from the 20-member set of *amino acids*.

Gene *expression* refers to the process by which genes are used to produce proteins. The mechanisms of gene expression are somewhat different for *prokaryotic* organisms, such as bacteria, that lack cell nuclei, and higher, or *eukaryotic*, organisms. We discuss only the differences that are germane to the task of finding genes.

The process of gene expression is illustrated in Figure 1. The first step in this process is called *transcription*, and involves the synthesis of an RNA molecule using DNA as a template. RNA that is eventually translated into protein is known as mRNA (for messenger RNA). The structure of RNA is very similar to DNA, except that it is composed from a slightly different alphabet of {`A, G, U, C`}, where `U` represents the ribonucleotide uracil. Each ribonucleotide of an RNA strand matches the DNA from which it was transcribed except that each `T` nucleotide in DNA is replaced with a `U` ribonucleotide in the RNA. (Actually, the synthesized RNA is complementary to one strand of the DNA and is identical – except for `T` → `U` substitutions – to the other strand. In our discussion, we follow biological convention and refer to the gene as being on the strand that is identical to the RNA.) The enzyme *RNA polymerase* is the molecule that transcribes DNA into RNA. RNA polymerase begins transcription after it binds to a regulatory *signal* called a *promoter* on a DNA molecule. In eukaryotic DNA, each gene is transcribed independently, and thus there is a promoter before every gene. In prokaryotic DNA, however, several consecutive genes may be transcribed into a single, continuous RNA molecule, thus there is not necessarily a promoter preceding each prokaryotic gene.

The *translation* process involves synthesizing a protein molecule using an mRNA strand as a template. A complex molecule called a *ribosome* performs the task of "reading" an mRNA strand and using its message to assemble a protein chain. Recall that amino acids

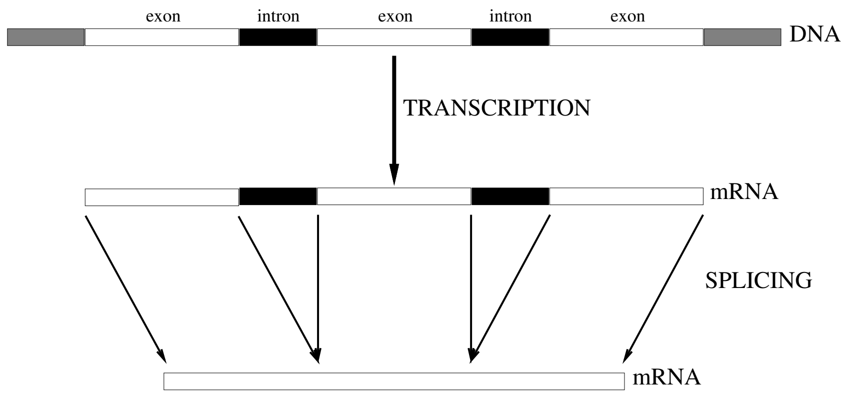

Figure 2: **mRNA splicing in eukaryotic organisms.** In higher organisms, or *eukaryotes*, parts of the mRNA chain are spliced out before translation occurs. The regions of a gene that are translated to protein are termed *exons*, and the regions that are spliced out are called *introns*.

are the building blocks of proteins. Each string of three consecutive nucleotides in mRNA encodes a single amino acid. The nucleotide triplets are called *codons*, and the mapping from codons to amino acids is called the *genetic code*. For a given strand of DNA, there are three different ways in which the nucleotides can be grouped into triplets: a given nucleotide can occupy the first, second, or third position in a codon. Only one of the three possible groupings is actually "read" by the ribosome; this grouping is termed the *reading frame* of the gene. As an analogy, consider a bit stream that contains a message encoded in ASCII. Such a bit stream has eight possible reading frames, and the correct frame must be found in order to decode the message.

In eukaryotic organisms, there is another significant step that occurs during gene expression. As illustrated in Figure 2, after RNA is transcribed, certain parts of the molecule are spliced out before it is translated to protein. Thus, genes in eukaryotes consist of alternating segments of *exons*, the nucleotide sequences that are expressed (translated to protein), and *introns*, the intervening sequences that are spliced out. The boundary points where splicing occurs are termed *splice junctions*.

The *genome* of an organism refers to the complete complement of DNA found in each cell of the organism. The human genome contains about 6 billion nucleotides and an estimated 100,000 genes. The genome is often called the blueprint for an organism since each gene is a plan for a protein, and proteins are the key building blocks of organisms. Unlike a blueprint, however, a large part of the genome does not contain such plans, but instead contains sequences that regulate the construction of the proteins, and sequences that may not have any useful function. Clearly, a fundamental problem in analyzing DNA sequences is to locate the genes (plans) in them. In the following sections we discuss machine-learning approaches to finding genes in DNA. Where applicable, we describe how the approaches differ for prokaryotic and eukaryotic DNA.

4 Search by Signal

There are two broad classes of computational approaches to finding genes in nucleotide sequences: *search by signal* and *search by content*. Search by signal locates genes indirectly by finding particular *signals* that are associated with gene expression. A signal is a localized region of DNA that performs a specific function, such as binding an enzyme. Search by content recognizes genes directly by identifying segments of DNA sequences that possess the general properties of coding regions. Search by content exploits knowledge of the differing statistical properties of coding and noncoding regions.

In this section we discuss machine learning approaches that have been applied to recognizing the biological signals that are involved in gene expression. Although our discussion focuses on search by signal as a method for locating genes in DNA, signal detection is an important problem in its own right. In order to understand the genome of a particular organism, it is necessary not only to understand the function of each gene, but also the mechanisms that regulate the expression of the gene. Many signals perform important regulatory functions by determining the conditions under which genes are expressed and the rate at which expression occurs.

There are several signals that can be identified in nucleotide sequences that are especially germane to identifying genes:

- transcription initiation sites (promoters),
- transcription termination sites (terminators),
- splice-junction sites,
- translation initiation sites (initiation codons),
- translation termination sites (stop codons).

The difficulty involved in identifying these sites varies considerably. Translation termination sites, for example, are trivial to identify. 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. Detecting translation termination sites, consequently, involves finding stop codons that occur *in-frame* in coding regions. A codon is said to be in-frame if it occurs in the reading frame of the gene. The tasks of identifying the other types of sites listed above, however, are reasonably complex. We discuss machine learning approaches to recognizing transcription initiation sites, translation initiation sites, and splice junctions.

As illustrated in Figure 3, the search-by-signal approaches that we describe all formulate their task as one of *classification*. Specifically, the tasks are defined in the following way: given a fixed-length "window" on a DNA sequence, determine if the window contains the signal of interest such that some identifiable feature of the signal occupies a particular position in the window. Once such a classifier has been trained, it can be used to locate the signals of interest by scanning its window along the length of the sequence.

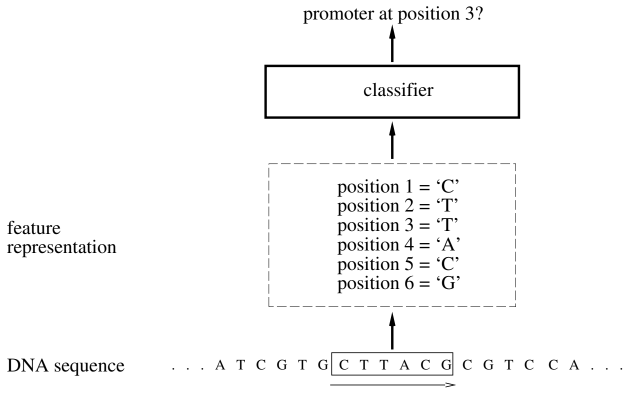

Figure 3: **Search by signal as a classification task.** Each instance is defined by the contents of a fixed-size window. The dashed box in the figure shows the window contents represented as feature-value pairs. The signals of interest (promoters, in this example) are located by scanning the classifier along the given sequence. A positive example occurs when a signal of interest is located in a specified position in the window (when a promoter begins at position 3, in this example).

4.1 Translation Initiation Sites

The first search-by-signal problem that we discuss is the recognition of translation initiation sites. Translation of mRNA to protein does not begin with the first nucleotide triplet of an mRNA molecule, but rather begins somewhere "downstream." In prokaryotic organisms, a single mRNA molecule may actually have several translation initiation sites since, in prokaryotes, consecutive genes may be transcribed into a single mRNA chain. Translation is usually initiated by the codon `AUG` which encodes the amino acid methionine. Identifying translation initiation sites, however, is more complicated than simply locating `AUG` codons.[1] The first `AUG` codon in an mRNA chain is not necessarily the initiation codon. Furthermore, in prokaryotes, translation sometimes begins with other codons, and `AUG` may also occur in the middle of a coding region. In prokaryotic organisms there is usually a sequence preceding the initiation codon that is complementary to the part of the ribosome that binds to mRNA. This sequence, called the *Shine-Dalgarno* sequence, is named for the biologists who proposed that mRNA and the ribosome bind to each other as part of initiation-site selection. Locating Shine-Dalgarno sequences can therefore aid in finding prokaryotic translation initiation sites. Recognizing them, however, is not straightforward since the nucleotides present in the Shine-Dalgarno region show considerable variation in actual translation initiation sites.

An early application of machine learning to molecular biology involved training *perceptrons* to recognize translation initiation sites in DNA of the bacterium *E. coli* [3]. As illustrated in Figure 4, a perceptron is an artificial neural network that has only one output unit and no hidden units. The input units of a perceptron represent features of the problem at hand. The input units shown in the figure, for example, represent three features: the nucleotides in particular window positions. Four units are used to represent each feature –

[1] Note that although translation initiation sites are actually features of mRNA, they can be recognized in DNA sequences since it is trivial to determine the RNA sequence that is transcribed from a given DNA sequence.

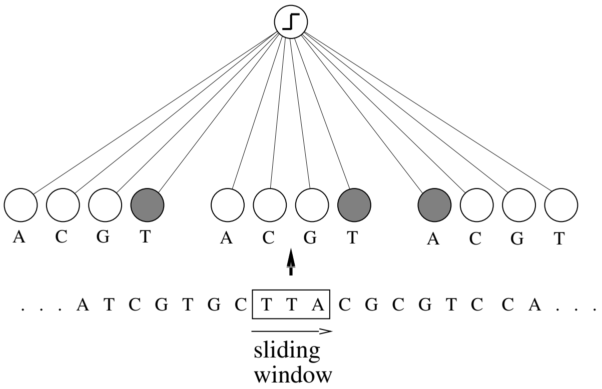

Figure 4: **A perceptron.** A perceptron is an artificial neural network that has a single thresholding output unit and no hidden units. In this figure, the input units represent the contents of the window. Shaded input units have activations of 1, the other input units have activations of 0.

---

one for each possible value that the feature can have. The state of a unit is represented by its *activation*, which is typically a real-valued number in the range [0, 1]. The activations of the input units are set to represent the feature values of a particular instance. Real-valued weights connect input units to the output unit. The activation of the output unit, for a given instance p, is calculated as follows:

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

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. A function, such as this one, that is used to compute activations is called an *activation function*. The output activation can be interpreted as the "answer" given by the perceptron. For example, an activation of `1` is typically interpreted as indicating a positive instance (e.g., translation initiation site), whereas an activation of `0` indicates a negative instance. Perceptron learning involves adjusting the network weights and threshold to maximize the number of training instances that are correctly classified. Specifically, several passes are made through the training set where the weights are updated for each instance in the following manner:

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

where t_{pi} is the *teaching signal*, or correct response, for instance p, and η is a step-size parameter that determines the rate of learning. A comprehensive introduction to neural-network learning techniques can be found elsewhere [4].

In training perceptrons to recognize translation initiation sites, Stormo and his colleagues experimented with windows that were 101, 71, and 51 nucleotides wide. As in Figure 4, they used four input units to represent each nucleotide in the window. The positive instances consisted of 124 known initiation sites, and the set of negative instances was 167 sites that were falsely identified as initiation sites using a rule-based technique. The positive instances were aligned so that the initiation codon for each instance occupied the same window positions.

Stormo et al. found that the perceptron with 101-nucleotide window generalized better than the others. Not surprisingly, the most significant weights were those connected to the units representing the initiation codon and the nucleotides in the Shine-Dalgarno region.

One way to think about the concept represented by a perceptron is as a matrix where the rows represent `A`, `C`, `G`, and `T`, and the columns represent positions within the window. Each element of this matrix is a number that represents the associated weight for a particular nucleotide occurring in a particular window position. In fact, it is common to find descriptions of such "weight" matrices in the biological literature. Another way to think about a perceptron's concept representation is as an $(n-1)$-dimensional hyperplane where n is the number of input units. A pattern of activation on the input units corresponds to a point in the n-dimensional space. The class predicted by the perceptron is determined by the side of the hyperplane that the point is on. A perceptron is thus able to accurately represent only concepts that are *linearly separable*; that is concepts for which the positive and negative instances can be completely separated by a hyperplane. Artificial neural networks that have *hidden units* are able to form more complex concept descriptions than perceptrons. Hidden units are able to transform the space defined by input unit activations into another space in which it is more profitable for the output units to make linear discriminations. Since the early work of Stormo et al., learning algorithms for such multi-layer networks have been developed (e.g., backpropagation), and these networks have been applied to recognize translation initiation sites and other signals.

4.2 Transcription Initiation Sites

As previously mentioned, transcription begins just downstream from where RNA polymerase binds to a promoter. Promoters thus provide another class of biological signal that is useful for locating genes in DNA. Several research groups have investigated using artificial neural networks to recognize promoters. One such group, Towell, Shavlik, and Noordewier have employed a novel approach, the KBANN algorithm, that combines neural network and symbolic learning [5].

The KBANN algorithm uses a set of approximately-correct, propositional rules to initialize the topology and weights of a neural network. After the network is initialized, ordinary neural-network learning techniques are used to adjust the weights. In a conventional network, the weights are initially assigned small random values, and a suitable topology is determined through experimentation. One of the contributions of the KBANN algorithm is that it provides a method for using problem-specific knowledge, in addition to training examples, during learning. Networks initialized using the KBANN algorithm 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 the task of recognizing promoters in *E. coli* DNA. Figure 5 shows the promoter rule set they used and an initial KBANN network. The input units of their network represent a window of 57 nucleotides. The positive instances (i.e., promoter sequences) were aligned so that the transcription initiation site for each occurred seven nucleotides from the right edge of the window. Noordewier, a biologist, derived an approximately-correct set of rules for recognizing *E. coli* promoters from the biological literature. This rule set identified two sets of sequence patterns that should occur about 10 and about 35 nucleotides upstream from

```
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".
```

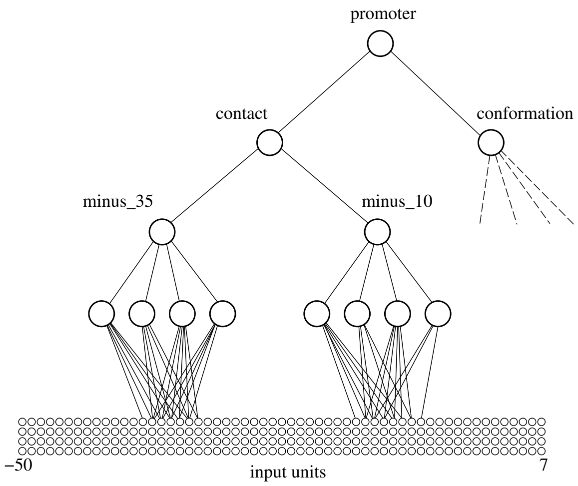

Figure 5: **The promoter rules and initial neural network.** The KBANN algorithm uses the approximately-correct rules to determine the topology and initial weights of the network. The rules are written here using a Prolog-like syntax. The notation `@-37 "cttgac"` is used to indicate that this rule is looking for the sequence `"cttgac"` thirty-seven nucleotides before the putative transcription initiation site. The character `x` in a sequence indicates that any nucleotide will match the rule at the given position.

---

where transcription begins. These two parts of the sequence, commonly referred to as the *-10* and *-35* regions in the biological literature, are the sites at which RNA polymerase binds to the DNA sequence. These sites are widely accepted as being the regions that provide the defining characteristics of promoters. Additionally, the rule set specified patterns for several other upstream regions whose significance is controversial. This latter group of rules, called the *conformation* rules, attempts to capture the effect of the helical structure of DNA on the spatial alignment of the *-10* and *-35* regions. 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. Through neural network training, however, the rules were refined so that they more accurately represented the essential characteristics of promoters.

Towell et al. compared networks initialized using the KBANN algorithm to conventional neural networks, decision trees, and nearest-neighbor classifiers (these approaches are discussed later in this article). They found that the KBANN networks generalized better than all of the other approaches. Their results indicate that the approximately-correct, task-specific rules used by KBANN assist neural-network learning by identifying some of the important problem features and significant relationships among them. Another interesting finding of their experiments was that the networks learned to discard the *conformation* rules during training. This result indicates that the *conformation* rules do not represent a salient aspect of promoters.

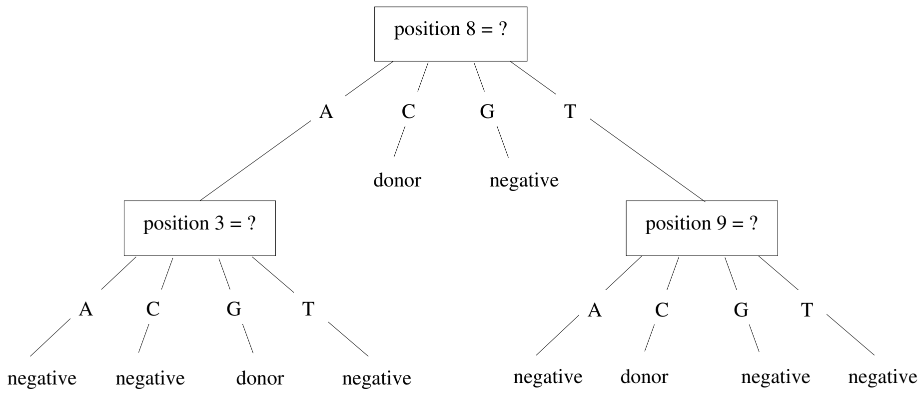

Figure 6: **A simple decision tree.** Each node in the tree represents a test applied to one of the problem features. Each leaf of the tree represents a predicted class; in this example the classes are *donor*, and *negative* (i.e., not a donor).

4.3 Splice Junctions

Because eukaryotic genes may contain introns, the problem of determining the extent of coding regions in eukaryotic DNA involves more than simply finding initiation codons. Recall that introns are sequences that are spliced out of mRNA before it is translated. Introns range in length from less than 100 to more than 1000 nucleotides. An important problem in eukaryotic sequence analysis is to identify the *splice junctions*, or boundary points where splicing occurs. Biologists commonly call the exon/intron borders *donors*, and the intron/exon borders *acceptors*. Identifying splice junctions is important because, in order to determine the protein produced by a gene, it is necessary to precisely demarcate the segments of the DNA sequence that are eventually translated.

Lapedes and his colleagues applied several machine learning approaches to the problem of recognizing splice junctions in human DNA [6]. In addition to artificial neural networks, they also used decision trees and k-nearest neighbor classifiers on this problem. Lapedes et al. used the ID3 algorithm [7] to induce decision trees. As illustrated in Figure 6, 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 possible outcomes of the test. With nominal features, for example, the test commonly results in a branch for each possible feature value. Each leaf of the tree represents a predicted class. Classification using a decision tree involves following a path from the root of the tree down to a leaf using the decisions made at each node to determine which branches are followed. Decision-tree learning is a recursive process that involves adding nodes to a tree until it sufficiently separates the training data by class. The learning algorithm selects a feature to branch on at each node, and then makes recursive calls to build subtrees for each created branch. All of the training instances are used to select the test at the root node, but smaller subsets of the training data are used to select the tests at subsequent nodes. Specifically, as the tree is constructed, it is also used to classify the training instances; only those training instances that reach a node are used to select the test at the node. The ID3 algorithm uses an information-theoretic measure to determine which feature is branched on at each node.

Lapedes and his colleagues also applied a k-nearest neighbor approach [8] to the problem of splice-junction recognition. The k-nearest neighbor method is a simple learning technique that does not require any training per se. The concept representation is simply the entire training set. In order to classify a new instance, the k "nearest" training instances are identified and the class label associated with the majority of these instances is the predicted class. The effectiveness of the approach is highly dependent upon the metric used to measure the distance between two instances. Lapedes et al. used a clever weighted Hamming distance. An ordinary Hamming metric defines the distance between two instances as the number of window positions in which the instances have different nucleotides. A weighted Hamming distance is calculated by associating a weight with each window position, where 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.

Lapedes et al. used windows of 11, 21 and 41 nucleotides to evaluate the neural network, decision tree, and nearest-neighbor approaches. Separate classifiers were trained to recognize the *donor* and *acceptor* classes. The instances were aligned so that the splice junctions were in the center of the window. For both donors and acceptors there is a pair of nucleotides that is *highly conserved* (`GT` for donors, and `AG` 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 were taken from known exons, were selected so that they also had either `AG` or `GT` in the center of the window. Selecting negative instances in this way prevented the classifiers from learning a trivial distinction such as: `AG` in the center of the window indicates *acceptor*.

Lapedes and his colleagues found that neural networks achieved higher test-set prediction accuracies than either decision trees or k-nearest neighbor classifiers. The *acceptor*-recognition networks correctly classified 91% of the instances in the test set, and the *donor*-recognition networks correctly classified 95% of these instances. Although the decision trees were not as accurate as the neural networks, they offered an advantage in that their concept representations are much more comprehensible. A decision tree can be easily transformed into a set of conjunctive rules. Lapedes et al. found that the rule sets obtained from the trained splice-junction trees were relatively small and biologically interpretable.

5 Search by Content

Unlike search-by-signal approaches, which look for specific functional sites in DNA, search-by-content methods identify genes by recognizing general patterns that occur in their nucleotide sequences. The objective of search-by-content methods is to identify the regions of DNA sequences that are translated to protein. 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 inter-genic noncoding regions, but also to distinguish introns from exons. There are three separate questions that search-by-content methods address: which regions are coding, and for a given region, which strand and which reading frame encode the protein. Recall that the *reading frame* of a gene refers to how consecutive nucleotides are grouped into triplets.

There are several properties that can be exploited to distinguish coding and noncoding regions. The overriding constraint that is placed on a coding region is that, by definition, it encodes a protein. The fact that some amino acids are used more frequently than others in proteins thus influences the nucleotide composition of coding regions. A second influence on the composition of coding regions is that there are different numbers of codons for different amino acids. This fact is due to the *degeneracy* of the genetic code. 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. A third influence on the nucleotide composition of coding sequences is that the codons that map to a given amino acid are not used equally in most organisms. This bias is termed the *codon preference* of the organism. Another simple constraint that is placed on coding regions is that they cannot contain stop codons. Finally, the function of a protein is largely determined by its shape, and its shape is partly determined by electrostatic interactions among neighboring amino acids. This influence means that some amino acids are more likely to be neighbors than others, and thus some codons are more likely to neighbor each other. The effect of all of these constraints is that the composition of coding regions is often significantly different from the nucleotide compositions of introns and inter-genic noncoding regions.

As with the search-by-signal approaches discussed in the previous section, the search-by-content methods that we describe make predictions based on a fixed-sized input window. By sliding the window of a trained classifier along a sequence, predictions can be generated for the entire length of the sequence.

5.1 Bayesian Approaches

Several search-by-content methods, including Staden and McLachlan's *codon usage method* [9], are based on Bayes' theorem. Given a window of nucleotides, their approach estimates the probability that each of the three reading frames of the strand encodes a protein. For a given sequence S occupying the window, the probability that frame i is coding (C_i) is calculated by:

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

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 $\frac{1}{3}$ for all three frames.) Each conditional probability, $P(S \mid C_i)$, is the probability that we would get the particular sequence S if we arbitrarily selected a coding sequence of the same length as S. These conditional probabilities are estimated by compiling a table of the frequencies of each codon in known genes of the organism. The frequency value for each codon is an estimate of the conditional probability that the codon occupies a given position in a sequence S, given that S encodes a protein. Staden and McLachlan make the simplifying assumption that the codons that compose a gene are independent of each other, and thus arrive at the following estimate:

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

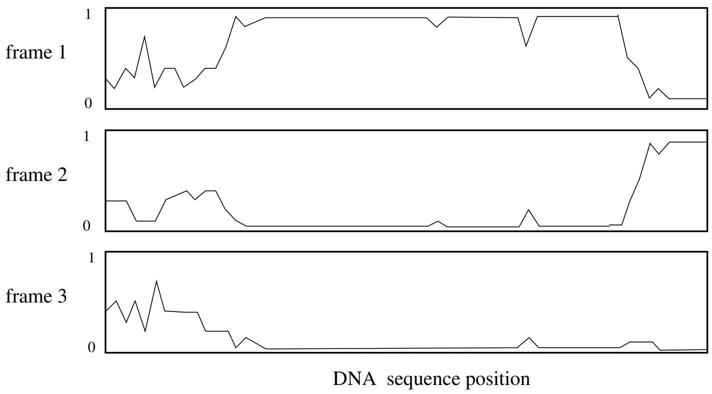

Figure 7: **Reading frame plots.** Each plot shows the predicted probability that the corresponding frame encodes a protein. The hypothetical set of signals in this figure shows the beginning of a protein coding region in frame 1, and a frameshift error that shifts the predicted reading frame to frame 2.

$S_i(j)$, in this equation, is the jth triplet in frame i in sequence S, and n is the number of triplets in frame i in S. This formula states that in order to determine the probability of finding sequence S in a coding region, calculate the joint probability of finding the individual codons of S in a coding 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 since the window may be positioned over a noncoding region, the approach still works well in practice, for reasons explained below. Moreover, it is straightforward to extend the *codon usage method* so that *noncoding* is considered as one of the possible hypotheses. Doing so requires estimating the prior probability of the *noncoding* hypothesis, as well as the conditional probabilities of each of the codons given the *noncoding* hypothesis. Estimating these probabilities is problematic for some species, however. Sequencing efforts often concentrate on areas that are dense with genes, thus there is a dearth of noncoding sequence data for some organisms. Additionally, it is sometimes difficult to ascertain that a stretch of putative noncoding DNA really does not contain a gene.

Typically, the *codon usage method* is used to generate a plot for each reading frame. As illustrated in Figure 7, each plot consists of a series of connected points that represent predicted probabilities for each window position. Coding-region boundaries are indicated by sharp changes in these plots. For example, the start of a coding region in the first frame would correspond to a steep increase in the predicted probabilities for the topmost plot. Even though the *codon usage method* assumes that a given sequence encodes a protein in one of its reading frames, noncoding regions can usually be detected because they tend to produce wildly fluctuating predictions, whereas coding regions result in consistently high probabilities.

The plots generated by the *codon usage method* have an additional use besides delimiting the boundaries and identifying the reading frames of coding regions. They are also able to detect *frameshift errors* in sequences. A frameshift error is not actually a feature of a DNA sequence itself, but instead is a laboratory error that occurs during the sequencing process. A frameshift error involves the mistaken insertion or deletion of a nucleotide in the sequence data. Because of the triplet nature of the genetic code, a frameshift error can have a devastating effect on the prediction of the amino-acid sequence translated from a given gene. Once the computed translation is out of frame, the predicted protein will bear no resemblance to the actual protein. Frameshift errors are often detectable, however, in plots such as those generated by the *codon usage method*. A frameshift error is indicated by a sharp drop in the plot of one reading frame accompanied by a steep increase in the plot of another frame.

A related statistical method, the Markov chain model, has also been applied to gene recognition [10]. This approach, like the Bayesian method, is based on computing the likelihoods of encountering a given sequence in each reading frame and in noncoding DNA. In the Markov chain approach, however, a DNA sequence is viewed as being generated by a state-based model. For example, Borodovsky and McIninch use a four-state model where each state corresponds to one of the four nucleotides. The prior probabilities of the states and the probabilities of the transitions are calculated from the training set. The likelihood of a given sequence is then calculated as the product of the initial state probability (i.e., the probability of the first element in the sequence) and the probabilities of successive state transitions. The statistics employed by a Markov chain model may describe *sequences* of transitions through several states: a kth order model uses statistics that describe transition chains that link $k+1$ states.

5.2 Neural Network Approaches

The assumption of the independence of codons in the window is perhaps the most problematic assumption made by the Bayesian *codon usage method*. Because interactions among neighboring amino acids partly determine the shape, and hence the function of a protein, neighboring codons are certainly not independent. Farber, Lapedes, and Sirotkin have shown that better coding-region predictions can be gained by taking into account the joint probabilities of neighboring codons [11]. The problem that they addressed in their experiments was to distinguish introns from exons. They demonstrated that perceptrons are able to outperform Bayesian approaches because they can account for some of the dependence between neighboring codons. They also showed that even better prediction accuracies can be obtained using a feature representation that explicitly represents adjacent codons.

In one experiment, Farber et al. compared the prediction accuracies of a Bayesian method to perceptrons with *sigmoid* activation functions. The sigmoid activation function, defined as $a_{pi} = \frac{1}{1+\exp(-\sum_j w_{ij}a_{pj}+\theta)}$, can be viewed as a continuous approximation of a threshold function. The Bayesian method they employed formulated the prediction task as a two-class problem: given a sequence, determine whether it occurs in an intron or an exon. This formulation of the problem assumes that the reading frame of the gene is known; the classifiers must simply distinguish introns from exons. Given a sequence S, probabilities are calculated in the following way:

$$P(Exon \mid S) = \frac{P(S \mid Exon) \times P(Exon)}{P(S \mid Exon) \times P(Exon) + P(S \mid Intron) \times P(Intron)}$$

The conditional probabilities $P(S \mid Exon)$ and $P(S \mid Intron)$ are estimated using the independence assumption and codon frequencies tabulated from sets of known introns and exons. The perceptrons use the same feature representation as the Bayesian approach. Specifically, this representation consists of 64 features that represent the frequency of occurrence of each codon. These features are represented using 64 input units, where the activation of each input unit is effectively a count of the number of times that the codon represented by the unit occurs in the window.

Farber et al. compared these two approaches using windows that ranged from 5 to 90 codons in length. The predictions of the perceptrons were significantly more accurate than those of the Bayesian method, especially with larger windows. The reason the perceptrons outperform the Bayesian approach is that they are not bound by the assumption of the independence of the codons. Farber, Lapedes and Sirotkin show that it is possible to set the weights of a perceptron by hand so that it calculates the same probabilities as Bayes Theorem under the independence assumption. These weights, however, are not optimal when the independence assumption is not true, as is the case for this problem. The training algorithm for perceptrons, however, is able to find optimal weights for the given training instances and feature representation, even when the assumption of independence is violated.

In a second experiment, Farber and his colleagues trained perceptrons that captured some codon dependencies in their feature representation. The features used in this experiment were all of the possible *dicodons*; that is, adjacent pairs of codons. Since there are 64 different codons, there are $64 \times 64 = 4096$ different dicodons. The perceptrons used in this experiment thus had 4096 input units, where each unit represented the frequency of occurrence of each dicodon. These perceptrons were trained using the same training sets and window sizes as in the first experiment. The generalization performance of these perceptrons was significantly better than the performance of the perceptrons that used only codon features. This result illustrates a common theme in machine-learning research: the ability of a learning system to find a good solution to a problem is highly dependent on the representation used for the features of the problem. Farber et al. found that even when hidden units were added to the networks that used the single-codon feature representation, the networks did not learn to represent dicodon frequency information well enough to match the performance of the networks that used the dicodon feature representation. Similarly, they found that using a feature representation of the individual nucleotides in the input window resulted in networks that did not generalize nearly as well as those that used a feature representation of codons.

Uberbacher and Mural have also applied neural networks to coding-region recognition in eukaryotic DNA [12]. Their *coding recognition module* (CRM) is a component in an automated sequence-analysis server called GRAIL [13]. The heavily-used GRAIL server accepts DNA sequences via electronic mail, analyzes them, and then returns e-mail messages describing the results of its analysis.

Uberbacher and Mural's research has also focused on finding a set of features that lead to good coding region predictions. The input features used by the *coding recognition module* are calculated by algorithms, called *sensors*, that evaluate seven different aspects of a given DNA sequence. The sequence characteristics measured by the sensors include such things as: 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 similarity to various repetitive patterns found in noncoding regions. There is a total of seven sensors, each of which provides a fair indication of the coding potential of the sequence being processed. During neural-network training, the CRM learns to weight the individual sensors and to recognize meaningful correlations in their values. Uberbacher and Mural evaluated their CRM using 19 human genes that were not in the training set. The CRM located 90% (71/79) of the long exons (more than 100 nucleotides) in these genes.

In addition to the CRM the GRAIL program also employs modules that predict splice junctions and translation initiation sites. An expert system with a blackboard control structure is used to assemble the predictions of the individual modules into coherent predictions of the location and intron/exon structure of genes.

5.3 Case-Based Approaches

Another AI approach to the problem of identifying protein-coding regions is a *case-based* approach. *Case-based reasoning* is a broad AI paradigm that involves several tasks, including indexing and retrieving cases stored in a memory. The indexing and retrieval aspects of case-based reasoning have been applied to gene recognition. The cases, in this context, are the nucleotide or amino-acid sequences of known genes. The case-based approach to gene recognition takes a new sequence, called the *query sequence*, and searches the case memory for similar sequences. A significant partial match between the query sequence and an element of the case memory can be interpreted 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 since, due to evolution, highly similar genes can be found in many different species. In addition to identifying potential coding regions, a matching gene in the case memory can also provide insight into the functionality of a newly-discovered gene. This is an interesting aspect of the case-based approach that distinguishes it from the other methods we have discussed.

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

Unlike the other gene-recognition approaches discussed in this article, the case-based approach does not use a fixed-size list of features to describe instances. Instead, a query sequence of arbitrary size is compared to case sequences of varying lengths. Although dynamic programming methods are able to find the optimal partial match between two different-sized sequences, these methods are too expensive to be used with large sequence databases. There are, however, several fast approximations to dynamic programming that are commonly used to search for similar sequences [14].

An issue that arises in constructing a case memory is deciding what constitutes a case.

Our discussion to this point has 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 domain has a specific function, and different combinations of domains (subroutines) give rise to different proteins (programs). An advantage of storing domains as cases is that they provide finer-grained units for predicting the functionality of query sequences.

How can a case memory of proteins and domains be assembled from a database of protein sequences? Hunter, Harris, and States have developed an *unsupervised* learning system that clusters related amino-acid sequences into domains and "families" of proteins [15]. Unlike the *supervised* learning methods which have been the focus of this article, unsupervised learning approaches are not told what the "correct" classes are, but instead they form their own class definitions. The goal of unsupervised learning is 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 et al. 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 developed a tool, called ClassX [16], that allows novel sequences to be matched against a case memory consisting of the clusters formed by their unsupervised learning algorithm.

6 Combined Methods

Although we have discussed how each of the search-by-signal and search-by-content problems is addressed in isolation, the most promising approaches to gene-recognition combine predictions of several different signals and coding regions. The previously mentioned GRAIL system is one such multi-strategy approach; another is the GeneId system [17]. GeneId predicts initiation codons, stop codons, donor sites, and acceptor sites and then assembles these predictions into possible genes. GeneId, like GRAIL, is publicly available as an e-mail server on the Internet.

The GeneParser system, developed by Snyder and Stormo [18], also integrates both signal and content predictions to identify introns and exons in DNA. GeneParser uses a dynamic-programming algorithm to predict the extent of individual exons and introns in a given DNA sequence. The dynamic-programming method employs two arrays which 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, are used to calculate the estimated likelihood values for these intron and exon arrays.

7 Conclusions

We have described two broad approaches to the problem of computationally identifying genes in DNA: search by signal and search by content. The search-by-signal approach locates genes by identifying the important DNA sites that are involved in gene expression. The search-by-content approach seeks to distinguish protein-coding regions from noncoding regions by recognizing the general patterns that characterize protein-encoding nucleotide sequences. We have described how a variety of inductive learning methods – neural networks, decision trees, Bayesian classifiers, and case-based systems – have been applied to both of these approaches. Additionally, we have briefly discussed several systems which combine search-by-signal and search-by-content methods to recognize genes.

Although machine learning approaches have shown great promise at recognizing genes in uncharacterized DNA, there is still much room for improvement. For example, it is very difficult to train signal classifiers that are both highly sensitive (i.e., predict few false negatives) and highly specific (i.e., predict few false positives). Similarly, search-by-content methods often fail to recognize short exons. We believe, however, that applying machine learning techniques to gene recognition is a fruitful enterprise for both molecular biologists and computer scientists. Biologists benefit from having effective, automated methods for analyzing sequence data. Machine learning researchers benefit from having important, real-world testbeds.[2] Because of these mutual interests, *computational biology* is a rapidly-growing field in its own right (e.g., see [19]). We expect that this marriage of computer science and biology will continue to advance the state of the art in both fields.

8 Acknowledgements

The research of the authors 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. The authors would like to thank Carolyn Allex, Rich Maclin, and Steve Gallant for providing helpful comments on a draft of this article.

[2]Data sets for several of the gene-finding problems discussed in this article are available by anonymous ftp from the UC-Irvine Repository of Machine Learning Databases and Domain Theories (ftp.ics.uci.edu).

References

[1] N. G. Cooper, editor. *Los Alamos Science, Number 20: The Human Genome Project*. Los Alamos National Laboratory, Los Alamos, NM, 1992.

[2] J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner. *Molecular Biology of the Gene*, volume I. Benjamin/Cummings, Menlo Park, CA, fourth edition, 1987.

[3] G. D. Stormo, T. D. Schneider, L. Gold, and A. Ehrenfeucht. Use of the perceptron algorithm to distinguish translational initiation sites in *E. coli*. *Nucleic Acids Research*, 10(9):2997–3011, 1982.

[4] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA, 1991.

[5] G. Towell, J. Shavlik, and M. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866. AAAI Press, Menlo Park, CA, 1990.

[6] A. Lapedes, C. Barnes, C. Burks, R. Farber, and K. Sirotkin. Application of neural networks and other machine learning algorithms to DNA sequence analysis. In G. Bell and T. Marr, editors, *Computers and DNA, SFI Studies in the Sciences of Complexity, vol. VII*, pages 157–182. Addison-Wesley, 1989.

[7] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[8] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[9] R. Staden and A. D. McLachlan. Codon preference and its use in identifying protein coding regions in long DNA sequences. *Nucleic Acids Research*, 10(1):141–156, 1982.

[10] M. Borodovsky and J. McIninch. Predictions of gene locations using DNA Markov chain models. In H. Lim, J. Fickett, C. Cantor, and R. Robbins, editors, *Proceedings of the Second International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis*, pages 231–248. World Scientific, Singapore, 1993.

[11] R. Farber, A. Lapedes, and K. Sirotkin. Determination of eucaryotic protein coding regions using neural networks and information theory. *Journal of Molecular Biology*, 226:471–479, 1992.

[12] E. C. Uberbacher and R. J. Mural. Locating protein coding regions in human DNA sequences by a multiple sensor – neural network approach. *Proceedings of the National Academy of Sciences*, 88:11261–11265, 1991.

[13] E. C. Uberbacher, J. R. Einstein, X. Guan, and R. J. Mural. Gene recognition and assembly in the GRAIL system: Progress and challenges. In H. Lim, J. Fickett, C. Cantor, and R. Robbins, editors, *Proceedings of the Second International Conference on Bioinformatics, Supercomputing, and Complex Genome Analysis*, pages 465–476. World Scientific, Singapore, 1993.

[14] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[15] L. Hunter, N. Harris, and D. J. States. Efficient classification of massive, unsegmented datastreams. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 224–232. Morgan Kaufmann, San Mateo, CA, 1992.

[16] N. L. Harris, D. J. States, and L. Hunter. ClassX: A browsing tool for protein sequence megaclassification. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, pages 554–563. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[17] R. Guigo, S. Knudsen, N. Drake, and T. Smith. Prediction of gene structure. *Journal of Molecular Biology*, 226:141–157, 1992.

[18] 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*, 21(3):607–613, 1993.

[19] L. Hunter, D. Searls, and J. Shavlik, editors. *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology.* AAAI Press, Menlo Park, CA, 1993.