Appears in *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology,* Bethesda, MD: AAAI Press (1993).

Protein Structure Prediction: Selecting Salient Features from Large Candidate Pools

Kevin J. Cherkauer
cherkaue@cs.wisc.edu

Jude W. Shavlik
shavlik@cs.wisc.edu

Computer Sciences Department, University of Wisconsin-Madison
1210 W. Dayton St., Madison, WI 53706

Abstract

We introduce a parallel approach, "DT-SELECT," for selecting features used by inductive learning algorithms to predict protein secondary structure. DT-SELECT is able to rapidly choose small, nonredundant feature sets from pools containing hundreds of thousands of potentially useful features. It does this by building a decision tree, using features from the pool, that classifies a set of training examples. The features included in the tree provide a compact description of the training data and are thus suitable for use as inputs to other inductive learning algorithms. Empirical experiments in the protein secondary-structure task, in which sets of complex features chosen by DT-SELECT are used to augment a standard artificial neural network representation, yield surprisingly little performance gain, even though features are selected from very large feature pools. We discuss some possible reasons for this result.[1]

Introduction

The problem of predicting protein secondary structures is the subject of much research. For quite some time, researchers in both molecular biology and computer science have attempted to develop rules or algorithms that can accurately predict these structures (e.g. Lim, 1974a, 1974b; Chou & Fasman, 1978; Qian & Sejnowski, 1988; King & Sternberg, 1990; Zhang, Mesirov, & Waltz, 1992). Researchers often make use of *inductive learning* techniques, whereby a system is trained with a set of sample proteins of known conformation and then uses what it has learned to predict the secondary structures of previously unseen proteins. However, the form in which the examples are represented is an issue which is often not well addressed. The performance of inductive learning algorithms is intimately tied to the representation chosen to describe the examples. Classification accuracy may vary widely depending on this representation, even though other factors are held constant (Farber, Lapedes, & Sirotkin, 1992; Craven & Shavlik, 1993), yet in most cases the learning systems are given only the names of the amino acids in a segment of protein with which to make their predictions. There is a wealth of other information available about the properties of individual amino acids (e.g. Kidera *et al.,* 1985; Hunter, 1991) that is ignored by these representations. This work tests the hypothesis that inclusion of this information should improve the predictive accuracy of inductive algorithms for secondary-structure prediction.

In order to test our hypothesis, we must address directly the question of representation choice. Our method, called "DT-SELECT" (Decision Tree feature Selection), chooses a small set of descriptive features from a pool that may contain several hundred thousand potentially salient ones. The method works by constructing a decision tree from a set of training examples, where the nodes of the decision tree are chosen from the pool. We use parallel processing to evaluate a large number of features in reasonable time. Once the tree is constructed, the features comprising its internal nodes provide a representation for use by other inductive learning algorithms. (Decision trees themselves can be used as classifiers, however we have found that on this problem they exhibit poor performance. This is one reason they are treated here primarily as a feature selection method, rather than an end in themselves.)

The exploration of very large feature spaces is a primary thrust of this work. It is our hope that such an approach will discover informative features which enhance the performance of inductive learning algorithms on this problem, either when used to augment more standard representations or, possibly, as self-contained representations.

Given this knowledge, our most surprising (and disappointing) discovery is that the ability to choose

[1]This work was supported by National Science Foundation Grants IRI-9002413 and CDA-9024618.

among thousands of features, many of which are quite sophisticated and domain-specific, does not appear to lead to significant gains on the secondary-structure task. Experiments that use features chosen by DT-SELECT to augment the standard input representations of artificial neural networks are detailed in the "Experiments" section, along with some speculations about their failure to improve the networks' classification accuracies. We also discuss the particular types of features included in the selection pools.

DT-Select

The DT-SELECT algorithm can be outlined as follows:

1. Construct a large pool of potentially useful features
2. Initialize an empty decision tree
3. Initialize the training set to all training examples
4. Select a feature (i.e. add a decision tree node):
   - 4a. Score all features in parallel on the current training set
   - 4b. Add the most informative feature to the decision tree (or terminate the branch if the stopping criterion is met)
   - 4c. Partition the current training examples according to the values of the chosen feature, if any, and recur (step 4) on each subset to build each subtree
5. Output the internal-node features as the new representation

The tree-building algorithm is essentially ID3 (Quinlan, 1986), except that the features examined at each iteration are drawn from the pool constructed in step 1, which may be very large and contain complex compound features. We describe the types of features currently implemented in the section titled "Constructed Features."

Scoring (step 4a) uses ID3's information-gain metric and is performed in parallel, with the feature pool distributed across processors. This metric gives a measure of the value of each feature in correctly separating the examples into their respective categories. The better a particular feature does at partitioning the examples according to class boundaries, the higher its information gain.[2] Repeated partitioning by several informative features in conjunction eventually yields sets of examples which contain only one class each. For this task, decision trees that completely separate the examples in this way actually overfit the training data. To alleviate this problem, we limit tree growth by requiring a feature to pass a simple χ^2 test before being added to the tree (Quinlan, 1986). This test, whose strictness may be adjusted as a parameter, ensures that included features discriminate examples with an accuracy better than expected by chance.[3]

Feature selection is accomplished efficiently by decision-tree construction, resulting in small sets of discriminatory features that are capable of describing the dataset. If we simply chose the n features from the pool with highest information gain, we would instead likely obtain a highly redundant feature set with poor discriminatory power. This is because many slightly different versions of the few best features would be chosen, rather than a mix of features which apply to different types of examples. Using decision trees for selection helps control this problem of highly correlated features, because at each selection step the single feature that best separates the remaining training examples is added to the tree. This provides a measure of orthogonality to the final set of features.

Almuallim and Dietterich (1991) offer a different feature-selection method, but its emphasis is on finding a minimal set of features that can separate the data, whereas our desire is to focus on the efficient discovery of informative—but not necessarily minimal—sets of relatively independent features. In addition, their minimization method is too computationally expensive to deal with the large number of features we wish to examine.

Because DT-SELECT searches a very large candidate set containing complex general and domain-specific constructed features, our hypothesis is that the selected features will capture important information about the domain which is not available in a more standard representation (e.g. one which merely specifies the amino acid sequence). The availability of these features may then enhance the abilities of other inductive learning algorithms to predict protein secondary structures.

The Feature Pool

The most important aspect of the DT-SELECT algorithm is its feature pool, which serves as a repository from which the tree builder chooses features. For conceptual and implementational simplicity, all features in the pool are Boolean-valued; thus, the decision tree that is built is binary.

Because of the complexity of the secondary-structure prediction task, it is difficult to know which features, from the myriad of possibilities, might be the most propitious for learning. The biological literature provides some clues to the kinds of features that are important for this problem (Lim, 1974a, 1974b; Chou & Fasman,

[2] We also intend to explore the use of Fayyad and Irani's (1992) *orthogonality measure,* which exhibits better performance on several tasks, as a substitute for information gain.

[3] We intend to replace this criterion with more a sophisticated overfitting prevention technique, such as the tree pruning methodology of C4.5 (Quinlan, 1992), in the near future.

Table 1: Partitionings of amino acids according to high-level attributes. Duplicate partitions are given identical numbers.

Partition	No.	Group	Amino acids
Structural partition	1.	Ambivalent	{A C G P S T W Y}
	2.	External	{D E H K N Q R}
	3.	Internal	{F I L M V}
Chemical partition	4.	Acidic	{D E}
	5.	Aliphatic	{A G I L V}
	6.	Amide	{N Q}
	7.	Aromatic	{F W Y}
	8.	Basic	{H K R}
	9.	Hydroxyl	{S T}
	10.	Imino	{P}
	11.	Sulfur	{C M}
Functional partition	4.	Acidic	{D E}
	8.	Basic	{H K R}
	12.	Hydrophobic non-polar	{A F I L M P V W}
	13.	Polar uncharged	{C G N Q S T Y}
Charge partition	4.	Acidic	{D E}
	8.	Basic	{H K R}
	14.	Neutral	{A C F G I L M N P Q S T V W Y}
Hydrophobic partition	12.	Hydrophobic	{A F I L M P V W}
	15.	Hydrophilic	{C D E G H K N Q R S T Y}

1978; Kidera *et al.,* 1985); it is entirely possible, however, that different combinations or variations of these may also prove valuable for the learning task.

Humans cannot be expected to analyze by hand extensive numbers of such features, yet this kind of search may yield valuable fruit for such a challenging problem. Therefore, one of our primary goals is to automate the process in order to search as large a space of features as possible. We use a Multiple Instruction-Multiple Data (MIMD) parallel machine, Wisconsin's Thinking Machines Corporation CM-5, to accomplish this. At each selection step during tree building, the information contents of all the thousands of features in the pool are evaluated in parallel with respect to the current training partition. This can be accomplished in only a few seconds even with thousands of training examples. Clearly this search is more thorough than is possible manually. The following subsections describe the features that comprise the pool.

The Raw Representation

In order to detail the types of features the pool includes, first we must briefly describe the raw example representation from which they are constructed. The proteins we use are those of Zhang, Mesirov, and Waltz (1992) and are provided as primary sequences of amino acids (AAs). There are 113 protein subunits derived from 107 proteins in this dataset, and each subunit has less than 50% homology with any other. Each position in a sequence is classified as either α-helix, β-strand, or random coil. For all of the experiments reported here, each possible 15-AA subwindow of a protein constitutes an *example*, for a total of 19,861 examples (one per residue). The overall task is to learn to correctly predict the classifications of the center AA of unseen examples, given a set of classified examples for training. Subwindow sections overhanging the end of a sequence are filled in with a special code. The same code is also occasionally present within the dataset's proteins themselves to denote ambiguous amino acids. Since some of the primary sequences are actually protein fragments, there may in fact be more AAs following the end of a sequence as given, so it is reasonable to represent these areas using the same code as for internal ambiguity. This follows the representation of Zhang, Mesirov, and Waltz (1992).

DT-Select has available the raw primary representation of each example. In addition, in order to tap AA physical and chemical property information which is not available from the primary representation alone, the implementation also has access to two other sources of information about amino acids. The first is a table of ten statistical factors from Kidera *et al.* (1985). These are small floating point numbers (ranging from -2.33 to 2.41), different for each amino acid, which summarize 86% of the variance of 188 physical AA properties. Since the value of each factor averages to zero across amino acids, we simply use zero as the value of all factors for unknown AAs.

The second collection of AA information is the knowledge of various partitionings of the set of amino acids into groups sharing common higher-level aspects. These are shown in Table 1. Though the table shows 20 labelled subgroups of AAs, there are only 15 unique partitions, and it is membership in these which the program uses. (The partitions are numbered in the figure such that duplicate partitions share the same number.)

Constructed Features

DT-Select constructs all features in the feature pool from the example information specified in the preceding section. Currently we have implemented several types of general-purpose and domain-specific features, some of which are quite complex. As mentioned earlier, all features have Boolean values.

Table 2 summarizes the implemented feature types.

Table 2: A brief summary of feature types. See the "Constructed Features" section for further details.

Name	Domain
Nominal	Single amino acid names
Unary Numeric	Single-factor comparisons
Binary Numeric	Two-factor comparisons
Unary Average	Single-factor average comparisons
Template	Salient multi-factor comparisons
Unary Cluster	Single amino acid group memberships
Binary Cluster	Paired amino acid group memberships

Of these, nominal, unary numeric, and binary numeric comparisons are relatively general-purpose. On the other hand, templates and unary and binary clusters are specific to the domain. Finally, average unary features may be viewed as both: though they were inspired by the ideas of Chou and Fasman (1978), they could easily be applied to any domain whose examples contain numeric attributes for which averaging makes sense. We describe each particular feature type in more detail in the following subsections.

Nominal Features This type of feature asks, "Does example position P contain amino acid A?" Since there are twenty AAs and one ambiguity code in our data, for 15-AA examples there are

$$21\ AAs \times 15\ positions = 315$$

nominal features. Note that these are often the only features a typical artificial neural network (ANN) is given as inputs for this problem.

Unary Numeric Features These features compare a particular statistical factor of the amino acid at a given example position with the neutral value of zero. While this is technically a binary comparison, we call these "unary numeric" features because only one statistical factor is involved. The possible comparisons are less-than, equal-to, and greater-than, all of which are performed relative to one of a set of user-specified positive thresholds. A factor is *greater than zero with respect to a threshold* if it is strictly greater than the threshold. Likewise, it is less than zero w.r.t. the threshold if it is strictly less than the negated threshold. Otherwise it is considered equal to zero. A typical set of thresholds is $\{0,\ \frac{1}{8},\ \frac{1}{4},\ \frac{1}{2},\ 1\}$.

An individual unary numeric feature asks, "Is factor F of the AA in example position P {less-than | equal-to | greater-than} zero w.r.t. threshold T?" where the comparison is one of the three possible ones. Thus, using five threshold values there are

$$10\ factors \times 15\ positions \times 3\ comparisons \\ \times\ 5\ thresholds = 2,250$$

potential unary numeric features. Calculations in the following subsections also assume five threshold values.

Binary Numeric Features Binary numeric features are similar to unary numeric ones, except that they compare statistical factors of amino acids in two example positions with each other. A feature of this type is of the form, "Is factor F_1 of the AA in position P_1 {less-than | equal-to | greater-than} the factor F_2 of the AA in position P_2 w.r.t. threshold T?" F_1 may be the same as F_2 as long as P_1 and P_2 are different. Similarly, P_1 and P_2 may be the same if F_1 and F_2 are different. Thus, there are

$$\begin{aligned}&\{\ ((10 \times 10)\ ordered\ factor\ pairs \\ &\quad \times \binom{15}{2}\ unique\ position\ pairs)\ when\ P_1 \neq P_2 \\ &+ (\binom{10}{2}\ unique\ factor\ pairs^4 \\ &\quad \times\ 15\ positions)\ when\ P_1 = P_2,\ F_1 \neq F_2\ \} \\ &\times\ 3\ comparisons \times 5\ thresholds \\ &\quad = 167,625\end{aligned}$$

possible binary numeric features.

Unary Average Features These are identical to unary numeric features except that instead of comparing a single statistical factor to zero, they compare the average of a factor over a given subwindow of an example to zero. To wit, these features ask, "Is the average value of factor F of the subwindow with left-most position P and width W {less-than | equal-to | greater-than} zero w.r.t threshold T?" Subwindows from width two to the full example width of 15 are allowed and can be placed in any position in which they fit fully within the example. Thus, there are 14 subwindows of width two, 13 of width three, and only one of width 15. Summing, there are

$$10\ factors \times \sum_{i=1}^{14} i\ subwindows \times 3\ comparisons \\ \times\ 5\ thresholds = 15,750$$

possible unary average features.

[4] The set of possible comparisons itself provides symmetry, alleviating the need to include *ordered* factor pairs in this case.

Table 3: Three- and four-place templates used.

Three-place templates	
● ● ●	Local interactions
● ○ ● ○ ●	β-strands
● ● ○ ○ ●	α-helices (hydrophobic triple)
● ○ ○ ● ●	α-helices (hydrophobic triple)
● ○ ○ ● ○ ○ ●	α-helices
● ○ ○ ● ○ ○ ○ ●	α-helices
● ○ ○ ○ ● ○ ○ ●	α-helices
● ○ ○ ○ ● ○ ○ ○ ●	α-helices (hydrophobic run)
Four-place templates	
● ● ● ●	Local interactions
● ● ○ ○ ● ●	α-helices (overlapping 1-5 pairs)
● ○ ● ○ ● ○ ●	β-strands (alternating), α-helices (overlapping 1-5 pairs)
● ○ ○ ● ● ○ ○ ●	α-helices (overlapping 1-5 pairs)
● ○ ○ ○ ● ○ ○ ○ ● ○ ○ ○ ●	α-helices (long hydrophobic run)

Template Features Template features are the first wholly domain-specific features we describe. They essentially represent particular conjunctions of unary numeric features as single features. A template feature picks out several example positions and performs the same unary numeric comparison with the same factor and threshold for all of them. The feature has value *true* if the test succeeds for all positions and *false* otherwise. We have thus far implemented three- and four-place templates. Because template features consolidate the information from several unary numeric features into one, they may have larger information gains than any of the unary numeric features would individually. This is important, because the patterns of AAs examined are restricted to ones which appear to operate in conjunction during protein folding, as explained in the next paragraph.

The three or four positions of a template feature are chosen such that they lie in one of a few user-specified spatial relations to one another (hence the name "template"). For example, triplets of amino acids which are four apart in the sequence may be important for α-helix formation, since they will all be on approximately the same "face" of the helix. Thus, the user may choose to specify a three-place template of the form "(1 5 9)." We represent this graphically as "(● ○ ○ ○ ● ○ ○ ○ ●)." This notation defines a spatial relationship, or *template*, among three amino acids; it is not meant to restrict a template feature to looking only at the first, fifth, and ninth amino acid in an example. On the contrary, the template may be "slid" to any position in an example, provided the entire template fits within the example. Thus, this particular template would also generate features that examine the second, sixth, and tenth AAs, as well as ones that access the seventh, eleventh, and fifteenth.

To summarize, a template feature asks, "Is factor F {less-than | equal-to | greater-than} zero w.r.t. threshold T for *all* AA's in template M with its left end at position P?" Table 3 gives graphic representations of all the templates used in these experiments, along with annotations as to their expected areas of value. Most of them were suggested by Lim (1974b, 1974b). Using the set of templates in Table 3, there are a total of

$$10\ factors \times 120\ template\ positions \\ \times\ 3\ comparisons \times 5\ thresholds = 18,000$$

template features.

Unary Cluster Features These are domain-specific features similar to nominal features, but instead of directly using the names of AAs, they check for membership in one of the groups, or "clusters," of related AAs given in Table 1. These features ask, "Does the AA in position P belong to cluster C?" There are

$$15\ clusters\ \times 15\ positions = 225$$

unary cluster features.

Binary Cluster Features Binary cluster features comprise all pairwise conjunctions of unary cluster features which examine different positions. That is, they ask, "Does the AA in position P_1 belong to cluster C_1 and the AA in position P_2 to cluster C_2?" where P_1 and P_2 are distinct. Thus there are

$$(15 \times 15)\ ordered\ cluster\ pairs \times \binom{15}{2}\ positions \\ = 23,625$$

binary cluster features.

Table 4: Feature pools for Tree 1 through Tree 4.

	Tree 1	**Tree 2**	**Tree 3**	**Tree 4**
Nominal	$\surd$	$\surd$	$\surd$	$\surd$
Unary Numeric	$\surd$	$\surd$	$\surd$	$\surd$
Binary Numeric	$\surd$	$\surd$	$\surd$	$\surd$
Unary Cluster	$\surd$	$\surd$	$\surd$	$\surd$
Binary Cluster	$\surd$	$\surd$	$\surd$	$\surd$
Unary Average			$\surd$	$\surd$
Template	All but ••• ••••	All but ••• ••••	All	All
Thresholds	$\frac{1}{4}$	$0\ \frac{1}{8}\ \frac{1}{4}\ \frac{1}{2}\ 1$	$\frac{1}{4}$	$0\ \frac{1}{8}\ \frac{1}{4}\ \frac{1}{2}\ 1$
Total Features	**60,990**	**208,290**	**64,890**	**227,790**

Experiments

In order to evaluate the worth of feature sets selected by DT-Select, we compared the classification correctnesses of ANNs that use a standard input representation to those of ANNs whose representations are augmented with these sets. The "standard" ANN input representation encodes only the particular amino acid sequence present in an example. The representations of *augmented* ANNs add to this the features chosen by DT-Select. (Preliminary experiments indicated that, for this particular task, augmentation of the standard feature set results in better performance than simply using the features chosen by DT-Select alone.) To the fullest extent possible, all experimental conditions except the different representations (and attendant network topologies) were held constant for the corresponding standard and augmented ANNs compared.

Cross-Validation

All experiments followed a ten-fold cross-validation (CV) paradigm. It is important to ensure that all the examples from a single protein subunit are either in the training *or* the testing set during all cycles of CV to avoid artificially overestimating correctness through effects of training on the testing set. Thus, the experiments were set up by first separating the 113 subunits into ten separate files. Nine of these are used for training and the tenth for testing in each of ten iterations, so each file is the testing set once. We created these files by first ordering the subunits randomly and then, for each subunit in the list, placing it in the file which at that time contained the shortest total sequence length. This method balances the desires for a completely random partitioning of the data and the obtainment of files of approximately equal size. The same ten files were used in all experiments.

The Networks

All networks were feed-forward with one layer of hidden units and full connection between layers. (We ran each cross validation experiment using networks having 5, 10, 20, and 30 hidden units.) Weights were initialized to small random values between -0.5 and 0.5, and we trained the ANNs with backpropagation for 35 epochs. For each epoch, the backpropagation code tracked correctness on a tuning set consisting of 10% of the training data. The weights from the epoch with the highest tuning-set correctness were used for the final, trained network. (The use of a tuning set makes it possible to control overfitting without using the testing set to choose the stopping epoch.) The output layers contained three units, one for each of α-helix, β-strand, and random coil, and the one with highest activation indicated a network's classification of an example.

The standard ANNs used a typical "unary" input encoding of the amino acids in an example (which corresponds to the 315 nominal features of DT-Select). Specifically, this encoding uses 315 input units: 21 for each of the 15 example positions. For each position, only one of the 21 units is on (set to 1.0) to indicate which amino acid (or the ambiguity code) is present. The remaining input units are set to 0.0. ANNs using an augmented representation have the 315 inputs of the standard ANNs plus additional binary inputs corresponding to the values of the features chosen by DT-Select.

To preserve the cross-validation paradigm employed, augmenting features for each fold of the CV had to be chosen separately using only the data in that fold's training set. Therefore, the ten augmented networks for a particular ten-fold CV run did not always have identical augmenting features or even the same number of features. This is unavoidable if contamination with information from the testing sets is to be avoided. However, the resulting network size differences were

Table 5: Average tree sizes (number of features) for Tree 1–Tree 4.

Decision Tree	Avg. Size
Tree 1	31.8
Tree 2	46.7
Tree 3	17.5
Tree 4	32.5

Table 6: Best test-set percent correctnesses (averaged over the ten folds) of the standard and augmented ANNS.

ANN Type	% Correct
Standard	61.5
Tree 1	61.6
Tree 2	61.2
Tree 3	61.9
Tree 4	61.0

small relative to the overall network sizes.

Augmentations

To explore the utility of representation augmentation by our method, we made four different attempts at network augmentation, using features chosen by DT-SELECT from differing pools of features with varying stopping criterion strictnesses. For each of these experiments, DT-SELECT built ten decision trees, one from each training set, using a fixed feature pool. For convenience, we will lump the first experiment's ten sets of augmenting features under the label "Tree 1." Likewise, we shall call the other sets "Tree 2," "Tree3," and "Tree 4."

The features in the pools used to build these trees are summarized in Table 4. The sizes of the resulting individual trees are given in Table 5. The differences in average sizes of the four sets of trees are due to varying the strictness of the stopping criterion used during decision-tree construction.

Results

The best test-set correctnesses (averaged over the ten folds) observed across the four different numbers of hidden units for the standard and augmented networks are given in Table 6. (There was little variation in correctness over the differing numbers of hidden units.) We see that the augmented networks unfortunately did not produce the performance gains we had anticipated achieving with DT-SELECT. The Tree 1 and Tree 3 networks did obtain slightly higher correctnesses than the best standard network, however the improvements are not statistically significant.

It is of some interest to see how well the original sets of decision trees themselves classify the data. This information is given in Table 7. It is evident that the neural networks do substantially better for this problem than the decision trees by themselves.

Table 7: Average correctnesses of the decision trees on the test sets.

Tree 1	Tree 2	Tree 3	Tree 4
55.9%	56.7	57.2	57.6

The use of the Wisconsin CM-5 was crucial in making these experiments possible, both because of its large memory and its parallel computing power. Our CM-5 currently has one gigabyte of main memory for each of two independent 32-node partitions. The largest of the experiments reported here required approximately 586 megabytes of this. Since precomputed feature values occupy most of this space, the mere availability of so much real (as opposed to virtual) memory alone adds substantially to tree-building performance by eliminating paging.

We were able to run all of the tree-building experiments in only a few hours of total CPU time on the CM-5. The longest-running (affected by both number of features and strictness of stopping criterion) of the four cross-validated tree-building experiments was Tree 2, which took approximately 3.35 hours on a 32-node partition of the CM-5 to build and test all ten trees. From this run we estimate a search rate in the neighborhood of 290 million feature values per second. Each decision tree node required about 12.8 parallel CPU seconds to construct.[5]

Figure 1 shows a decision tree constructed for one of the cross-validation folds of the Tree 3 experiment. This tree obtained 57.7% correctness on its training set and 56.0% on its testing set. The tree constructed in the actual experiment had 13 internal nodes, but due to the strict stopping criterion used to build it, several subtrees had leaves of only a single class. In the figure we have collapsed each such subtree into a single leaf for simplicity. The simplified decision tree will thus make the same classifications as the original. However, it is important to note that the features in the collapsed subtrees were retained as inputs for the neural networks, as they do provide information on example classification that may be of use to the ANNs.

Discussion

It was surprising to us to find that the addition of apparantly salient domain-specific features of great sophistication to the input representations of ANNs does not lead to gains in classification performance. The

[5]System software development for the CM-5 is currently ongoing. These performance analyses should be taken as rough estimates only and may vary with different versions of the system software.

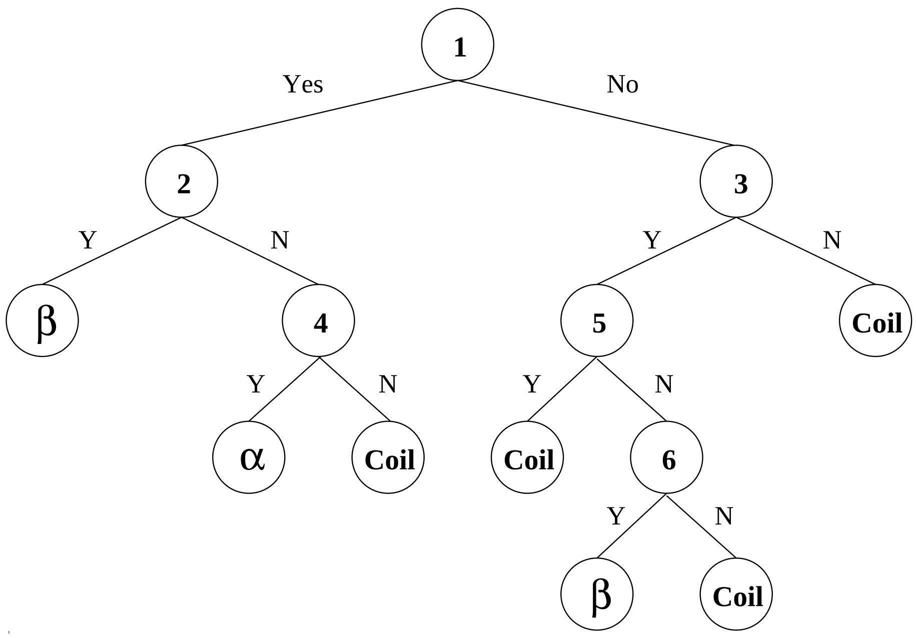

Figure 1: A sample decision tree for protein secondary-structure prediction. Leaves are labelled with classifications. Recall that the window has 15 positions, and we are attempting to predict the secondary structure of position 8. Internal nodes (labelled "1" through "6") refer to the following Boolean-valued features:

1. Is the average value of Kidera factor 1 (roughly, anti-preference for α-helix) over positions 5–13 $< -\frac{1}{4}$?
2. Is the average value of Kidera factor 3 (roughly, β-strand preference) over positions 5–11 $> \frac{1}{4}$?
3. Is the average value of Kidera factor 3 over positions 6–9 $> \frac{1}{4}$?
4. Is the average value of Kidera factor 1 over positions 7–9 $< -\frac{1}{4}$?
5. Is the average value of Kidera factor 1 over positions 6–9 $> \frac{1}{4}$?
6. Is the average value of Kidera factor 1 over positions 1–14 $> \frac{1}{4}$?

reasons for this remain unclear at this time, although several possibilities exist.

First, it could be that the ANNs themselves are capable of deriving similar types of features on their own using their hidden units. If this is true, it demonstrates that the power of backpropagation to develop useful internal representations is indeed substantial, given the complexity of the features available to DT-Select during tree construction. This implies that perhaps the implementation of even more complex types of salient features which backpropagation is *not* capable of deriving itself is necessary for the use of DT-Select to yield performance improvements.

Second, it is possible that we have not yet implemented the best types of features for augmentation. There are always new types of features one can think of adding to the system; for instance, binary average features which examine two factors in two subwindows, or templates which average factors over the AAs they examine. Indeed, we have observed that, in general, adding new features tailored to the secondary-structure prediction task, such as the cluster and average features, tends to displace the more general types of features from decision trees out of proportion to their numbers. In Tree 2, the cluster-type features account for approximately one third of all features chosen, though they constitute only about 10% of the features in the pool. Even more remarkable, the addition of the unary average feature in Tree 3 and Tree 4 resulted in more than half of the features of each of these tree sets being of this type, though they comprise only about one quarter of the features in the pool for Tree 3 and about 7% of those in Tree 4's pool. (All features in Figure 1's collapsed tree are unary average features.) This indicates that these features are higher in information "density" than the more general-purpose features they displace, yet, strangely, we do not see great advances in either decision-tree or augmented-network correctnesses when such features are added to the pool.

Thus, a third possible explanation for the lack of observed gain is that, for ANNs of this form, the 315 inputs of the standard unary encoding actually capture all the information such networks are capable of using. This is an interesting hypothesis in itself, and the reasons behind it, if it is true, would be worth uncovering—especially considering the effects of input representation on DNA coding-region prediction mentioned earlier (Farber, Lapedes, & Sirotkin, 1992;

Craven & Shavlik, 1993). Our standard ANNs were used only as controls, and the learning parameters were not extensively optimized. (The augmented networks used the same parameters as the standard ones.) However, though the complete system of Zhang, Mesirov, and Waltz (1992) attained a considerably higher correctness, the ANN component of their system alone achieved an accuracy comparable to that of our ANNs. Some researchers postulate an upper bound of 80–90% correctness on this problem using local information (Cohen & Presnell, personal communication, 1991), but it is possible that techniques more sophisticated than single feed-forward ANNs are needed to get beyond the low-60% range when using datasets of the size currently available.

Conclusion and Future Work

We introduced the DT-SELECT system, which attempts to select, from large candidate pools, compact feature sets that capture the relevant information about a problem necessary for successful inductive learning. The method operates by building decision trees whose internal nodes are drawn from the pool of features. It offers an efficient way to select features that are both informative and relatively nonredundant. We tested the utility of this approach by using the sets of features so selected to augment the input representations of artificial neural networks that attempt to predict protein secondary structure. We observed no significant gains in correctness, leading us to surmise three possible explanations for this negative result. Regardless of which of these, if any, is the correct one, we believe the method yet exhibits potential for extension in this domain and for application to other problems, both inside and outside molecular biology.

Immediate future work will include the addition of other problem-specific features to our implementation, the replacement of the χ^2 stopping criterion with a more modern pruning methodology, and experimentation with a replacement for the information-gain metric. We have also begun to test DT-SELECT on the problem of handwritten-character recognition, with preliminary results already showing correctness gains over a standard encoding. We will soon begin testing the method on the DNA coding-region prediction task as well.

A more general issue we intend to explore is the use of selection algorithms other than decision trees. We have performed a few initial experiments with two other algorithms, one which builds Foil-like rules (Quinlan, 1990) to describe the individual example classes and another which applies statistical independence tests to select features which are largely orthogonal, but more work is needed to determine the strengths and weaknesses of different feature-selection approaches.

References

Almuallim, H., & Ditterich, T.G. (1991). Learning With Many Irrelevant Features. *Proceedings of the Ninth National Conference on Artificial Intelligence,* Vol. II (pp. 547-552). Anaheim, CA: AAAI Press/The MIT Press.

Chou, P.Y., & Fasman, G.D. (1978). Prediction of the Secondary Structure of Proteins from their Amino Acid Sequence. *Advances in Enzymology,* 47, 45-148.

Craven, M.W., & Shavlik, J.W. (1993). Learning to Predict Reading Frames in *E. coli* DNA Sequences. *Proceedings of the Twenty-sixth Hawaii International Conference on System Science* (pp. 773-782). Maui, HI: IEEE Computer Society Press.

Farber, R., Lapedes, A., & Sirotkin, K. (1992). Determination of Eucaryotic Protein Coding Regions Using Neural Networks and Information Theory. *Journal of Molecular Biology,* 226, 471-479.

Fayyad, U.M., & Irani, K.B. (1992). The Attribute Selection Problem in Decision Tree Generation. *Proceedings of the Tenth National Conference on Artificial Intelligence,* (pp. 104-110). San Jose, CA: AAAI Press/The MIT Press.

Hunter, L. (1991). Representing Amino Acids with Bitstrings. *Working Notes, AAAI Workshop: AI Approaches to Classification and Pattern Recognition in Molecular Biology,* (pp. 110-117). Anaheim, CA.

Kidera, A., Konishi, Y., Oka, M., Ooi, T., & Scheraga, H.A. (1985). Statistical Analysis of the Physical Properties of the 20 Naturally Occurring Amino Acids. *Journal of Protein Chemistry,* 4, 1, 23-55.

King, R.D., & Sternberg, J.E. (1990). Machine Learning Approach for the Prediction of Protein Secondary Structure. *Journal of Molecular Biology,* 216, 441-457.

Lim, V.I. (1974a). Algorithms for Prediction of α-Helical and β-Structural Regions in Globular Proteins. *Journal of Molecular Biology,* 88, 873-894.

Lim, V.I. (1974b). Structural Principles of the Globular Organization of Protein Chains. A Stereochemical Theory of Globular Protein Secondary Structure. *Journal of Molecular Biology,* 88, 857-872.

Qian, N., & Sejnowski, T.J. (1988). Predicting the Secondary Structure of Globular Proteins Using Neural Network Models. *Journal of Molecular Biology,* 202, 865-884.

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

Quinlan, J.R. (1990). Learning Logical Definitions from Relations. *Machine Learning,* 5, 239-166.

Quinlan, J.R. (1992). *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Zhang, X., J.P. Mesirov, D.L. Waltz (1992). A Hybrid System for Protein Secondary Structure Prediction. *Journal of Molecular Biology,* 225, 1049-1063.