Protein Structure/Function Classification Using Hidden Markov Models

Thesis by

Moira E. Regelson

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1997

(Submitted June 4, 1997)

Acknowledgements

Many thanks to Jerry Solomon and Dan Meiron for their invaluable assistance through the years. I would also like to thank Tom Hou and Scott Fraser for their interest in my work. Finally, thanks to my parents for their support and encouragement, and particularly to my mother for her editorial advice.

Abstract

Three-dimensional protein structures can be divided into classes in which proteins demonstrate high similarity of structure. While full and accurate determination of a protein's three-dimensional structure from its amino acid sequence is not feasible at this time, methods seeking to determine this full structure would be aided by a priori information about the sequence's overall structural class. We have utilized Hidden Markov Models on sequences from the SWISS-PROT database in an attempt to determine the structural class of a protein given only its primary amino acid sequence.

Varying representations of the amino acid sequences and the accuracy with which the models using these representations differentiate between classes give some insight into the chemical and physical properties which are significant in the protein folding process. In addition, some representations of the protein sequence can illustrate the redundancy in the protein alphabet and others can capture structural class information with reduced computational requirements. Real vector representations provide an analogy to the problem of speech recognition.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Biological background: the protein folding problem

Given a protein, the primary question is "what does it do?" The answer to this question lies in the details of the protein's structure. The amino acid sequence defining a protein will quickly fold into a specific and compact three-dimensional structure called the "native" fold. This structure contains regions whose shape and composition allow specific types of interactions with other molecules. Through these interactions, a protein performs its function.

Determination of the structural details of a protein's fold is a complex and difficult task. Tools for the direct examination of folded proteins exist but are difficult to apply and do not always yield the desired detail [11]. Finding the amino acid sequence of the protein is relatively easy, whether by direct sequencing or decoding of DNA. We would like to be able to look at this sequence and draw some conclusions about the protein's shape and, hence, its function.

The amino acid sequence contains all the information necessary to define the protein's native fold, but we do not know how to extract this information. In theory, we should be able to model the folding process and determine the native fold directly from the sequence. In practice, however, the many attempts made to model the biology, chemistry and physics of this folding process have met with limited success. Most attempts to determine a protein's native structure from its sequence fail to find this structure with much accuracy [18]. The complex forces driving the protein folding process have not been well modeled at this time.

1.2 Structural classification of proteins

While we cannot yet accurately determine the specific placement of each amino acid within the protein's three-dimensional structure, we may at least be able to determine the protein's general structural or functional class. Information about the general structural class of a sequence can provide a rough idea about the role of the protein and may assist methods which search for more specific three-dimensional structure. In this work, we have used Hidden Markov Models with several different representations of amino acid sequences in an attempt to distinguish between different structural classes of proteins. We also hope that the nature of the representations will give us some insight into the forces driving the protein folding process.

1.2.1 Utility of classifying sequences by three-dimensional structure

Though determination of the general folding class of a protein would not entirely solve the protein folding problem, it could provide assistance to methods which model the physical and chemical interactions of the process. For example, some models use a Monte Carlo (or other type of search) approach to search for appropriate low energy folds [3, 4, 17, 25, 30] and a preliminary process which determined the general structural class of a protein sequence could aid these models by significantly reducing the search space. This reduction in the search space could increase the chances of finding the structure to which the protein would naturally fold.

Knowledge of folding class can assist attempts to determine the full native fold of a protein and can also help approaches seeking to determine sub-structures (or "secondary" structures) within the native fold. Protein structure is traditionally considered at the primary, secondary and tertiary levels. The amino acid sequence itself defines the primary structure of a protein, and secondary structure refers to three-dimensional sub-patterns which form along the length of the amino acid string, such as alpha helices and beta strands. Tertiary structure refers to the superstructure formed by the secondary structural elements. Knowledge of the tertiary structural

class gives some information as to what secondary structures are likely to be formed by a given sequence and can therefore help secondary structure prediction methods [26].

1.2.2 Approaches to identification of structure

Three-dimensional structure prediction from sequence has been attempted at both the secondary and tertiary levels. Sequence similarity is most commonly used in attempts to identify the structural class of a sequence at both levels. This has some utility, but the first problem encountered is that of alignment between the sequence (or segment of a sequence in the case of secondary structure) to be classified and the sequences belonging to a known structural class. The second, more relevant to this work, is the existence of sequences which fold to the same general structure but with very low sequence similarity to other members of the class.

Profiles reflecting the statistics of sequence similarities for classes of proteins have been generated and used to "score" sequences [5, 14, 12, 16, 20, 21, 23, 29, 31]. While specific profile construction varies widely, these methods generally rely heavily on sequence alignment and attempt to identify regions of sequence similarity (common sub-strings of amino acids). The underlying assumption of such approaches is that sequence similarity very probably reflects structural similarity. However, these approaches fall short when faced with sequences demonstrating very little amino acid similarity but much structural and functional similarity.

Prediction of tertiary structural class is complicated by the wide variety of definitions of tertiary structural class. One common definition is based on the arrangement of secondary structure elements within the protein. This definition divides proteins into four or five rudimentary structural classes (α, β, $\alpha + \beta$, α/β and "multi"). Other definitions in this broad vein exist, defined with respect to a particular aspect under study. For example Nishikawa, Kubota and Ooi [22] categorized protein structures into four groups: intracellular enzymes, intracellular non-enzymes, extracellular enzymes and extracellular non-enzymes. They were able to correctly identify approx-

imately 66% of the sequences they studied. Alternately, tertiary structural classes may be defined by considering a combination of structure and function. Varying definitions of tertiary classification can make comparison of results somewhat difficult, but the inherent ambiguity of the problem would tend to increase acceptance of any consistently good results on consistent definitions of classes.

One of the most successful methods of prediction of tertiary structural class to date uses only the "acid composition" of a sequence. This simply refers to a vector in which the n^{th} entry is the percentage of times amino acid n appears in the sequence. This is a very different approach from the commonly used profile methods mentioned above, which specifically track each position in the sequence. Chou and Zhang [7, 8, 9] have worked with amino acid composition and used comparison with class statistics to assign a sequence to one of the above rudimentary structural classes. They have found the correct classifications of over 90% of unknown sequences, although there has been some dispute over the selection of the test sets for these experiments [15].

While amino acid composition may or may not play the primary role in protein folding, it certainly has a strong correlation to fold type. Different types of secondary structure elements show strong preferences for certain types of amino acids; thus, composition of a protein sequences could easily correlate to tertiary structure based on secondary structure composition. A neural network approach has been applied to the amino acid composition of sequences (and yet another definition of structural classes) with an average accuracy rate of 87% [13].

Other very successful methods of tertiary structural classification have involved the use of Hidden Markov Models (see 1.3.2). These entail hypothesizing that sequences from a structural class are the result of a random process.

1.3 Structural class as a random process

1.3.1 "Decoding" protein sequences into structural classes

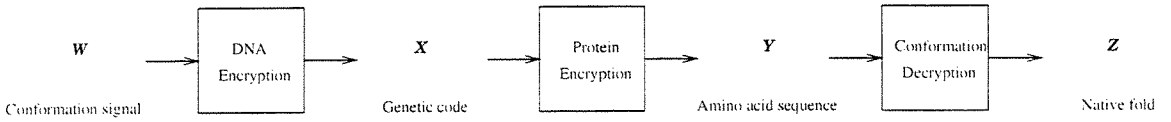

Figure 1.1: Structural Class encryption and decryption. Note that "Protein encryption" is equivalent to DNA decryption

If we look at the amino acid sequence of a protein as a code for structure, we can consider protein folding as the decoding process. In fact, there are two analogous steps to the process of producing a folded protein from DNA sequence information. In translating the DNA sequence into an amino acid sequence, we have a more transparent decoding process because it consists of well defined code words ("codons") corresponding to specific amino acids. The folding decoding process, as we postulate it here, consists of "decoding" a protein's amino acid sequence into a particular general structure.

We hypothesize that each structural class of proteins corresponds to a random process, X. The signals from this process generate, by passage through a hypothetical channel, the genetic code. The genetic code, in turn, produces a sequence of amino acids by passage through another hypothetical channel.

In decoding segments of DNA, we find great redundancy in the codons for certain amino acids and, similarly, a great many amino acid sequences correspond to a single general structural class. When the ultimate objective of the process is to generate a protein with a particular structure/function, we must have redundancy to compensate for decoding errors, first at the genetic level and then again at the amino acid sequence level.

While there are similarities between the generation and decoding of DNA and of protein sequences, the rules for generating amino acids from codons have been well documented. By contrast, the folding process appears to rely on much longer and more complicated code words, the coding rules of which we have not yet identified. Even without a full understanding of the rules of encryption, we can at least attempt to determine some essential characteristics of the sequences belonging to a common structural class. In particular, we attempt to model the random process

which generated the sequences by using a Hidden Markov Model.

1.3.2 Approaches to classification using Hidden Markov Models

The variations in amino acid sequences corresponding to a structural class clearly indicate a need for a non-deterministic model of sequence generation. We should note that the variability of the sequences does not necessarily mean that the randomness originates with the process itself. We could as easily assert that the process is deterministic and that noise and decoding errors in the hypothetical channels introduce the random factor. However, in this work we choose to consider amino acid sequence generation as the direct result of a random Markov process, X', disregarding the DNA generation and decoding phases. The decoding of DNA into an amino acid sequence is a relatively well understood and deterministic process which does not illuminate the underlying random process.

Hidden Markov Models have been used extensively in pattern recognition applications, notably for speech recognition problems [24]. They model random processes and can recognize patterns with variations as being generated by the same random process. Since we would like to find common ground for widely varying protein sequences, Hidden Markov Models offer a means of modeling the random process generating these sequences.

Hidden Markov Models have been used previously to attempt to identify protein structural classification from amino acid sequence data. White *et al.* have carefully constructed models [26, 28] to reflect the secondary structures known to be present in certain tertiary structures and then used filtering techniques to determine, for a given sequence, the likelihood that each model produced the sequence. Their test data was drawn from the Brookhaven Protein Data Bank (PDB).

Krogh *et al.* used a similar approach and chose to consider the nodes of a Hidden Markov Model as representing positions in secondary structural elements. Unlike White *et al.*, they did not attempt to hand-tailor the parameters of the model and

used traditional training methods instead. The primary goal of their approach appears to have been sequence alignment and search for similarity. They achieved good recognition of the Globin structural class, as indicated by higher scores by Globins on the Globin model than sequences from other classes displayed [19]. Baldi *et al.* have used a modified version of the Krogh approach with an alternate training procedure for the model, with similar results. This approach is designed for on-line training and requires somewhat less computation time [1, 2]. Both of the latter groups used data from the SWISS-PROT database.

1.4 Extension of Hidden Markov Model approaches

While the above methods essentially test sequences for membership in a particular structural class, we choose to define a classification scheme. We have achieved success rates of up to an average of 94.8% in distinguishing between the Globin, Calcium-Binding and Kinase structural classes and a "Random" class which represents sequences from other classes. While other methods using Hidden Markov Models [1, 2, 19] have achieved comparable results in detecting members of the Globin and Kinase classes, they used only full, twenty letter alphabet representations of amino acid sequences. We have considered both this full alphabet and several reduced symbolic alphabets. In addition, we have considered real vector representations of sequences which require significantly less computation, while still successfully detecting up to an average of 94.58% of sequences in the above classes.

1.4.1 Maximum likelihood classification

In this work, we use performance on a Hidden Markov Model as a criterion for a classification scheme. Instead of considering variations between scores on a single model, we consider the performance of a single sequence on multiple models, each trained for a different class. We then assign the sequence of unknown structure to the class corresponding to the model with the highest probability of having generated the sequence.

1.4.2 Alphabet variations

The alphabet used in previous applications of Hidden Markov Models to the protein folding problem consists of symbols representing the twenty amino acids which constitute the protein's primary structure. We have used this alphabet, while also considering alternate representations of the protein sequence.

Alphabet reduction

While still representing a sequence symbolically, we can explore the essential information contained in the sequence. We look at various groupings of the amino acids primarily based on their properties of hydrophobicity, polarity and charge. In this way, we reduce the size of the alphabet composing the sequence. Note that with this reduced alphabet we may find increased similarity between sequences within a structural class.

If we can reduce the size of the alphabet without significantly affecting performance of the model, we will have placed a bound on the entropy of the random process which generated the sequence by displaying redundancy within the set of symbols used to generate the code words (sequences). And, in a more practical sense, we may have identified the essential properties which the amino acids represent at each position.

Real vector representations

In addition to alternate symbolic representations of the primary sequences, we have considered real vector representations of protein sequences. We have primarily considered processed property profiles in an analogy to signal processing, but we have also considered amino acid composition because of its significantly good, if somewhat controversial, results in prediction of the rudimentary structural classes.

We generated "property profiles" by replacing each amino acid by its value in a table for the property being modeled. This provides a real vector representation of an amino acid sequence with respect to a biochemical property. We treated each

property profile as a signal from the random process and computed the Fourier power spectrum, which we then passed to the Hidden Markov Model.

An additional real vector representation is given by considering the amino acid composition of a sequence. In this representation we reduced the sequence to a 19-dimensional vector representing the percentage of each amino acid within the protein sequence. Like the property profile, this vector represents a "property," but this is a property of the entire sequence, not of the individual amino acids.

We already know that Hidden Markov models using symbolic representations of protein sequences (with the full alphabet) can differentiate between structural classes. By using real vector representations, we hope to establish that this faster approach is equally effective. In addition, we hope to be able to evaluate the significance of various properties in the folding process.

Chapter 2 A Hidden Markov Model approach to protein structure/function classification

2.1 General Hidden Markov Model definition

Hidden Markov Models (HMMs) work on the general principle that we are modeling a Markov process and that this process can be considered to be in one of a discrete set of states at any time. A change in "time" generally occurs with a transition between states. Relating the process to a set of actual observations means simply assigning symbol emission probabilities to each state.

Consider a path, q, within the model consisting of a sequence of states $q[t]$. Since we assume we have a Markov process, we assume the states are memoryless and that the probability of transition from state $q[t]$ to state $q[t+1]$ depends only on state $q[t]$ and no preceding states. We also assume that this transition probability is independent of time, so that if $q[t] = i$ and $q[t+1] = j$, the transition probability from state i to state j is fixed, regardless of the time index at which the system passes through state i.

We can completely specify such a model by a number of states (N), an alphabet (S), an initial probability distribution on the states (π), a state transition probability matrix (A), and a symbol emission probability matrix (B). A path through the states of the model probabilistically generates a sequence of observations, o in which $o[t]$ is generated by some state $q[t]$ in the given set of states.

π is defined for each state of the model and $\pi[n]$ denotes the probability that a path through the model will start in state n. $A[n][n']$ is defined for all states n and n' and represents the probability of the next state being n', given that the current

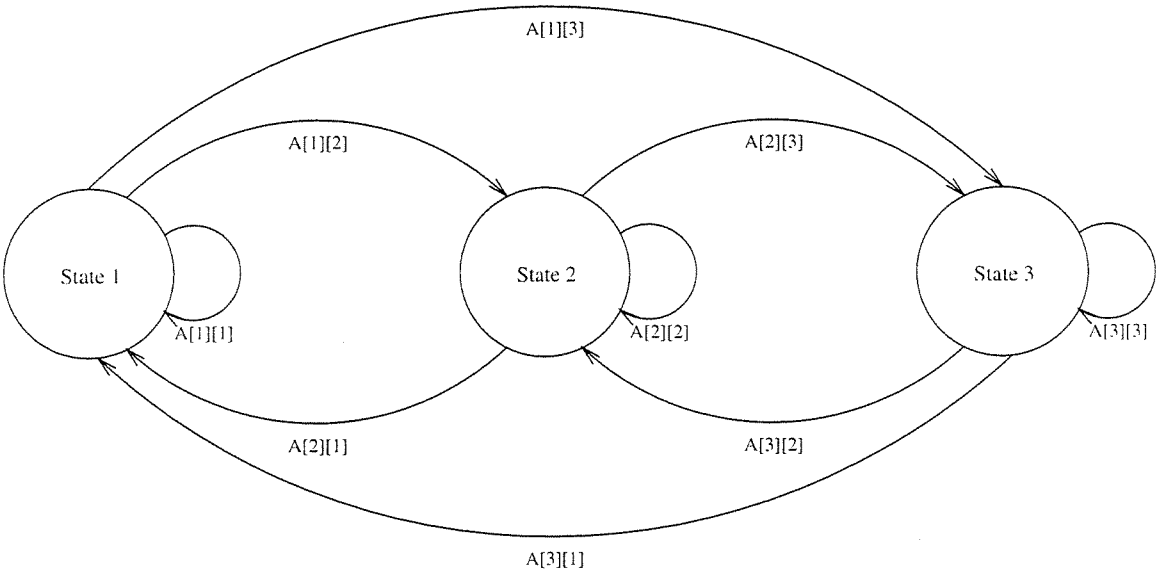

Figure 2.1: A generalized three-state Hidden Markov Model, allowing all possible state transitions

state is n. The "hidden" aspect of the HMM is that $B[n][k]$ (the probability of seeing symbol k given that the system is in state n) may be non-zero for all k and all n. In this case we cannot be sure which state generated which symbol.

2.2 Application of Hidden Markov Models to protein sequences

2.2.1 Protein sequences as time series

HMMs are useful for modeling time series or other sequences of observations which can be considered as the product of a Markov process, and to a certain extent we can consider protein sequences in this way. We can consider the random process generating proteins of a particular structure as passing through a set of states, each emitting an amino acid. However, in the three-dimensional structure of a protein amino acids not adjacent in sequence may come in contact. Since there may be correlations due to these interactions, we cannot consider protein sequence generation as a strict Markov process. We can sidestep this question somewhat by considering

a left-to-right HMM in which transitions are never made to lower index states. This allows us to use the general idea of a Hidden Markov Model but still places some restrictions on position within the model for each time step. In this way we can hope to capture probabilities specific to position.

2.2.2 The protein alphabet

The fundamental alphabet of protein sequences is, of course, twenty symbols representing the twenty amino acids. Previous constructions of HMMs for protein classification use only this full alphabet, but we have also considered various groupings of amino acids by biochemical properties. We have primarily considered the properties of hydrophobicity, polarity and charge, but we have also tested some other common groupings of amino acids. These variations do not affect the construction of the model except for definition of alphabet size.

2.3 Specific model structure

We denote a protein's amino acid sequence with the vector o and hypothesize that it was generated by a left-to-right Hidden Markov Model.

A left-to-right HMM is defined by zero transition probabilities to states with a lower index, i.e. $A[n][m] = 0$ if $m < n$. The model consists of a sequence of nodes, each consisting of three states. To parallel Krogh *et al.*'s work, we refer to these states as *Match*, *Insert* and *Delete* . In addition, we introduce an *End* state to which the process goes after the last symbol in a sequence.

The *Match* states are the main line of the model. *Insert* states allow the process to remain at a node while the *Delete* states allow the process to skip a node. This terminology stems from Krogh *et al.*'s objective, which was primarily multiple sequence alignment. In Baldi's work, *Match* is replaced by "Main." However, if we conceive of the sequence generation process as attempting to generate some sort of archetypal sequence but affected by corrupting random influences, the term "*Match*" still applies. If a sequence lacks a certain component which tends to be characteristic

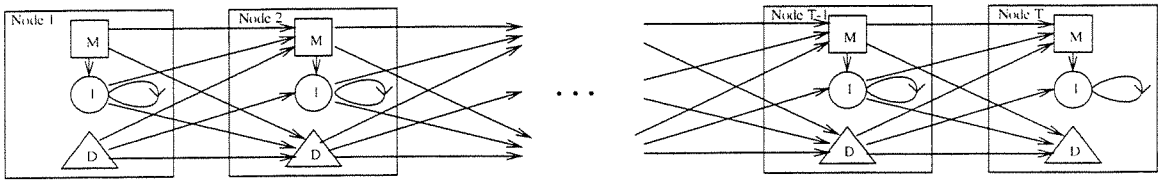

Figure 2.2: Allowable transitions between states of the model

of a structural class, we would say the process was in the *Delete* state at that position, whereas if extraneous amino acids appear which have no relevance to the structure, we would hypothesize that the process had been in the *Insert* state at that point.

Essentially, we are considering a three-state HMM, but we have duplicated this model T times to allow position variations. Transitions are still from and to those fundamental three states, but some transitions move the process forward a node. The final result is a Hidden Markov Model with $3T$ states, plus one for *End*. This model allows us to keep the probability of transition between states in this general sense fixed over "time," while still allowing necessary variations relating to position (time).

Notation

To more effectively represent the structure of this model, we use a slight modification of the conventional notation. A "state" in the global structure of the model will be represented by n_s where n is the node and s is the sub-state within the node. Henceforth, "state" will refer to the sub-state within a node.

2.3.1 Flow within the model: tracking the time step versus the node

We choose to define the "time step" in this context as the position in the amino acid sequence. This is a departure from the common definition of a time step for a Hidden Markov Model in which the t^{th} time step corresponds to the t^{th} transition made in the model. In this case the presence of the *Delete* state, which generates no symbol, makes that definition impractical. We also cannot define time step to correspond to the node of the model because it is theoretically possible that no amino acid in

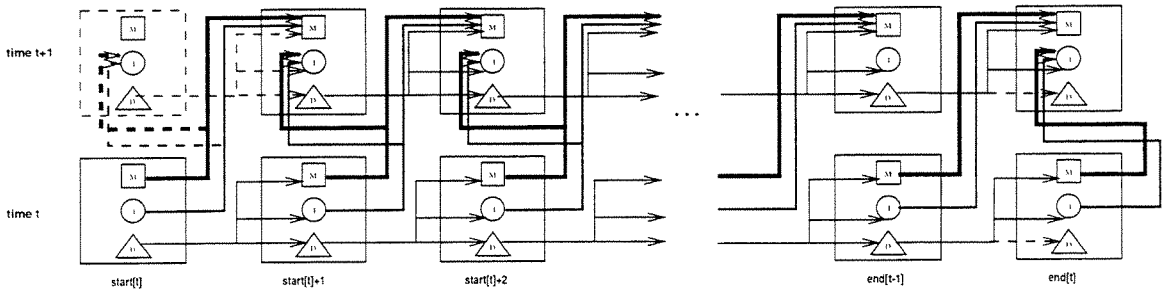

Figure 2.3: Transitions between nodes and time steps. Dashed lines indicate possibility that the start (or end) of the alignment windows for time t and time $t+1$ may be the same. In particular for $t < \frac{M}{2}$ (or $t > T - \frac{M}{2}$)

the entire sequence was emitted by a particular node. Because of the existence of the *Delete* state, we must consider alignment of the sequence to the model and by defining time step in this manner, we avoid introducing the additional considerations of alignment of the time step to the sequence.

To apply this convention about the time step, we require that any transitions from symbol producing states involve an increase in the time index. Transitions from the non-symbol producing *Delete* state remain within the same time step. Thus we may have a set of transitions associated with each time index t, beginning with a number of *Delete* states and ending with the state producing observation t.

2.3.2 Alignment to the model

The question of alignment arises in this approach, but in contrast to similarity searches, the question is of alignment to the model as opposed to other sequences. We could assert that the number of insertions to a given point in the sequence is, with high probability, the same as the number of deletions to that point. However, to allow greater flexibility, we have allowed for the possibility that the t^{th} node in the model may not have generated the t^{th} amino acid in a sequence ($o[t]$). This increases computation time by a factor on the order of the square of the alignment window size, but given variation in sequence length between members of the same structural class, it seems impossible to neglect the alignment question.

Having decided that the t^{th} amino acid in a protein sequence defines the t^{th} time

step, what node most probably generated this amino? Given the existence of both *Insert* and *Delete* states, there is a fair chance that the observation at that time step t was generated by node t. In attempting to determine which node generated amino acid t, we look within a window of size M centered at t for the most probable node. In particular, to be certain that we do not exceed the edges of the model, the alignment window ranges from $\max(1, \min(T, t - \frac{M}{2}))$ to $\min(T, t + \frac{M}{2})$. Though insertions tend to be somewhat more probable than deletions, with a wide enough window we can still hope to overlap the node which most probably generated $o[t]$ (as defined by generating $o[t]$ on the highest probability path through the states of the model, computed over all paths which might have generated o).

Considering the window as defined above affects the training of the Hidden Markov Model. We need only compute the training variables associated with observation t for nodes n in the window and not outside it, since we have constructed the window on the assumption that there is virtually zero probability that a node outside the window generated observation t.

2.3.3 Multi-domain sequences

The alignment window ends at the last node of the model for single domain sequences but may extend onto a concatenated version of the model for multi-domain sequences. For some structural classes, proteins within the class may consist of a sequence of domains, each conforming to the structure which defines the class. In general, attempts at classification of sequences have examined domains one at a time, considering them as individual sequences. In this model, however, we have considered the entire sequence at once, dividing it into segments only for training logistics.

In order to determine the probable length of each domain for these classes, we looked at histograms of length of sequences from the SWISS-PROT database. From these we determined approximate domain length ranges for a single domain from the Globin and Calcium-Binding classes. We rejected as fragments any sequences with length shorter than the minimum single domain length indicated. The histograms

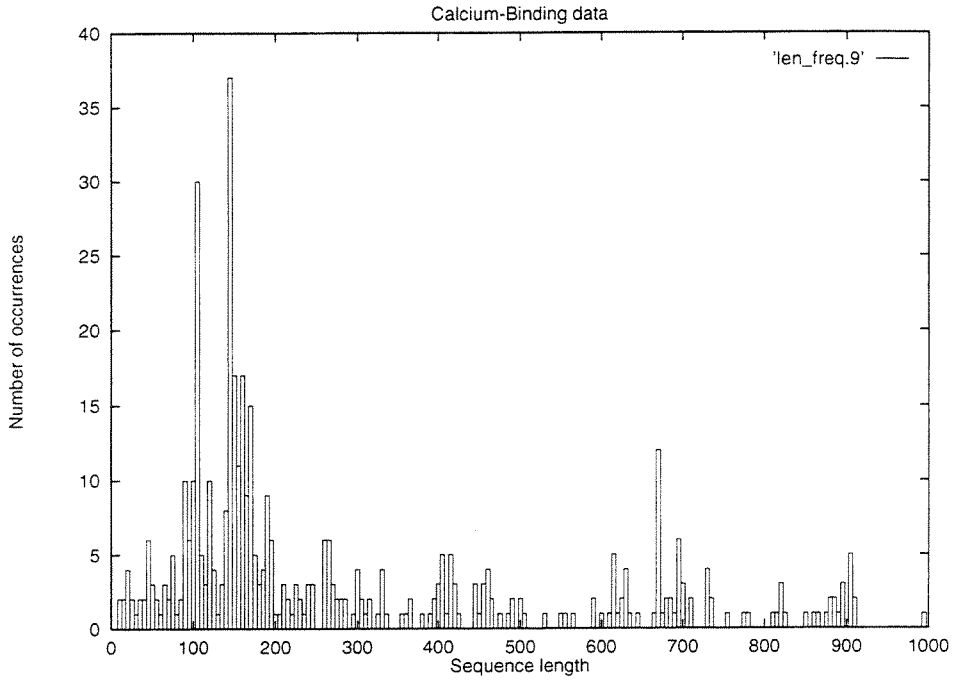

Figure 2.4: Sequence length distribution for Calcium-Binding sequences

display peaks roughly around integer multiples of the average of the domain ranges chosen.

There appear to be a large number of both single and double domain Calcium-Binding sequences, which makes determination of single domain length difficult. We have set the single domain length range to [70,130].

Note that, while there are very few occurrences of what we would call multi-domain Globin sequences, their lengths cluster together at regular intervals. We have chosen the domain range to be [115,170].

2.3.4 Selection of model length and window size

Given the minimum and maximum single domain lengths, we select a model length and window size to allow the last amino acids of the shortest and longest possible single domain sequences to be generated by the last node of the model. In particular for multi-domain sequences, we choose window size

$$M = \text{maximum domain length} - \text{minimum domain length}$$

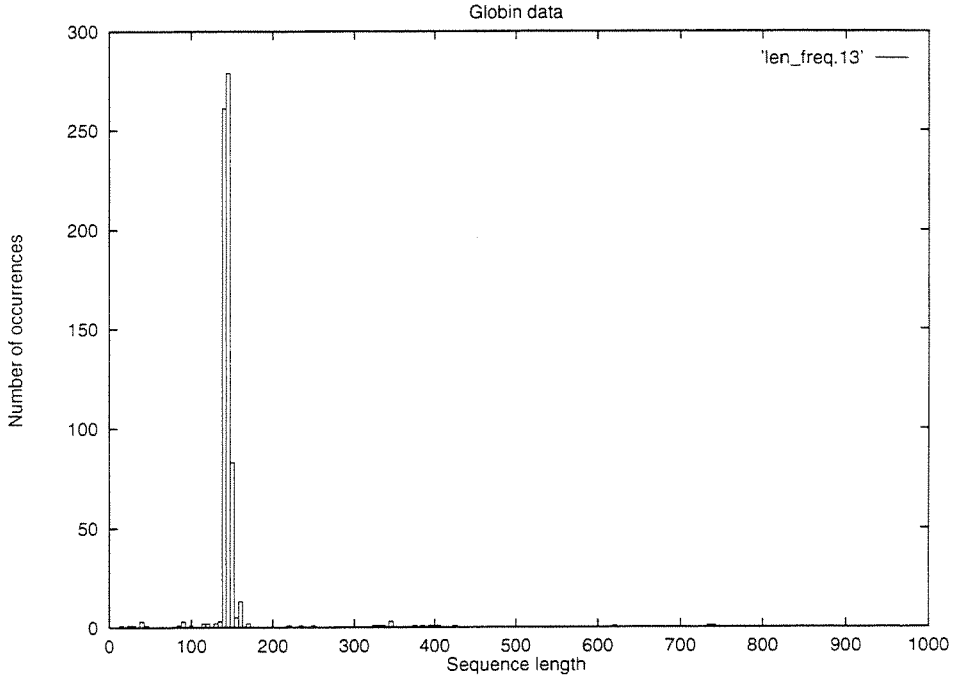

Figure 2.5: Sequence length distribution for Globin sequences

and model length

$$T = \text{minimum domain length} + \frac{M}{2} = \text{maximum domain length} - \frac{M}{2}$$

(with slight alterations if max - min is not even).

Other constant quantities associated with the model and derived from the above are the arrays

$$start_t = \max(1, \min(T, t - \frac{M}{2}))$$

and

$$end_t = \min(T, t + \frac{M}{2})$$

which define the edges of the window of nodes to examine for each time t. Note that $start_t$ as defined above insures that the entire window will lie on the model, even for exceedingly long sequences.

The window size is a fixed property of the model and is constant (except at the edges of the model) for every time t, however there is very likely another construction of window size, perhaps one varying with time step, that would work as well with

somewhat greater efficiency.

One alternative under consideration is to determine the sequence-independent probability of being at a node at a given time step, then to calculate the average node, $\overline{n}$, for that time step and, finally, to assign the window to be $[\overline{n} - 3\sigma, \overline{n} + 3\sigma]$, capturing the most probable set of nodes. However, if we constructed the window this way, we might miss updating nodes which it would be more appropriate not to ignore. We have not tested results with different window constructions, but it might be useful to do so. Given the wide range of single domain lengths, the window size determined by the fixed window length method may be quite large relative to sequence length, which has a great impact on computation time. With a large and fixed window, the window size may introduce computational factors on the order of the model length.

The alignment window of the "Random" class and other single domain classes

A "Random" structural class was constructed of protein sequences from SWISS-PROT not belonging to the Globin or Calcium-Binding groups. Since the Random class, by definition, lacks structural similarities, maximum and minimum domain lengths as determined above would not make sense. Thus, we simply restricted the lengths of sequences chosen and treated the entire class as consisting of single-domain sequences. The sequences ranged in length from 75 to 250 and the above convention for window size would be prohibitively large, so instead we used a fixed window size. We set this size (fairly arbitrarily) to 60. Similarly, sequences in the Kinase class do not exhibit repetitive patterns along their sequences; for these, as well, we fixed the window size to 60. In order to allow the longest sequence in the training set to have been generated by the model, we set

$$T = \max(seq\_len) - \frac{M}{2}.$$

Impact of time step and window conventions on model parameters

Combining the time step convention with the observation window restricts construction and update of certain transition probabilities. To remain consistent with the above definition of the time index, we must disallow transitions to the final *Delete* state within the window associated with time t. If we allowed a state sequence to pass through that state, no symbol could have been produced for the index t, as any transitions to symbol producing states would necessarily take us out of the window.

There is an exception to this rule at the final node of the model when considering multi-domain sequences, whose alignment window for later times may extend beyond the end of the model.

2.3.5 Initialization of the model

Initially, we set π and the state transition matrix, A, to constant values with higher probability of being in or going to the *Match* state than to the *Insert* or *Delete* states.

Symbol emission probability matrix

For the discrete alphabet case, the initial symbol emission matrix is set by considering the statistics of the training set. In particular, we compute the average number of times symbol m appears at position n in the sequences in the training set and use that as the initial probability of seeing m at n_s for $s \in \{Match, Insert\}$.

Thus,

$$B[n_s][m] = (\frac{1}{n\_seq}) \sum_{i=1}^{n\_seq} \delta(o_i[n], m)$$

where $\delta(o_i[n], m) = 1$ if the n^{th} amino acid in the i^{th} sequence in the training set corresponds to symbol m and 0 otherwise. As a convention, we consider the lack of a symbol in the *Delete* state to be a "*skip*" symbol and say that $B[n_{Delete}][skip] = 1$ for all nodes n, at all times. Since $B[n_s][.]$ is a probability distribution on the symbols in the alphabet, $B[n_{Delete}][m] = 0$ for all actual symbols m in the alphabet.

2.4 Evaluation of model performance: the Viterbi algorithm

For any given sequence, there exist multiple paths through the states of the model which could have generated the sequence. The *Delete* states within this path will, of course, not appear because the *Delete* state produces no symbol. For the purposes of examining paths through the model, we define a new state transition matrix A' in which the *Delete* states do not appear.

2.4.1 Construction of the A' matrix

Within the matrix A in our left-to-right model, the only possible transitions from node n go to either n or $n+1$. In particular, the transitions $Match{->}Insert$ and $Insert{->}Insert$ from node n go to node n, and all other transitions from node n go to node $n+1$. Therefore, in $A[n_s][next\_n_{s'}]$, we can infer $next\_n$ from n, s, and s'. However, we can construct a state transition probability matrix which represents transitions from node n to any higher-index node within a window by incorporating transitions to and from *Delete* states. In particular,

$$A'[n_{Match}][n_s] = 0$$

$$A'[n_{Match}][(n+1)_s] = A[n_{Match}][next\_n_s]$$

for $s \in \{Match, Delete\}$ because the transitions $Match- > Match$ and $Match- > Delete$ necessarily move to the next node. Similarly,

$$A'[n_s][n_{Insert}] = A[n_s][next\_n_{Insert}]$$

$$A'[n_s][(n+1)_{Insert}] = 0$$

for $s \in Match, Insert$ because those transitions stay within the node.

For steps of size greater than one node, we construct the transition probability

based on the state sequence we must have had to see a step of that size between two symbol producing states. If we start in n_s, where $s \in Match, Insert$ (i.e., a symbol-producing state) and we do not see a symbol until state $n'_{s'}$, then the process must have been in the *Delete* states for each node between n and n'. Thus, we have a transition from n_s to $(n+1)_{Delete}$, a sequence of transitions from *Delete* to *Delete*, and finally a transition from $(n'-1)_{Delete}$ to $n'_{s'}$. The probability of this sequence of transitions is given by:

$$A'[n_s][n'_{s'}] = A[n_s][(n+1)_{Delete}] \ (\Pi_{\nu=(n+1)}^{n'-2} A[\nu_{Delete}][(\nu+1)_{Delete}]) \ A[(n'-1)_{Delete}][n'_{s'}]$$

with the center term omitted if $n' = n+2$.

2.4.2 Construction of π'

In addition to eliminating the *Delete* states from the state transition matrix, we must also consider the vector $\pi[s]$, which represents the probability of starting a path in state s of the first node. When we eliminate the *Delete* states, we need a new vector $\pi'[n_s]$ which represents the probability that the first observed symbol was generated by n_s. Clearly,

$$\pi'[1_s] = \pi[s]$$

and, for subsequent nodes $n \in [start_1, end_1]$,

$$\pi'[n_s] = \pi[s] \ (\Pi_{\nu=1}^{n-2} A[\nu_{Delete}][(\nu+1)_{Delete}]) \ A[(n-1)_{Delete}][n_s]$$

with the center term omitted if $n = 2$.

2.4.3 Determination of best path through model

Given the above transition probability matrix, we can determine the best path through the model which could generate the observation sequence. This path is defined by a sequence of states, q, where we assume that state q_t generated the symbol $o[t]$. The

Viterbi algorithm is a dynamic programming approach to the problem of determining the best path of the many possible paths q. For each time, node and state, we calculate the highest possible probability path to that node and state at that time. The "best" path ends at the node and state which has the highest probability path to it at time $t = seq\_len$. Thus, in addition to storing the probabilities at each node and state of the model, we must store the paths to which they correspond in order to back-track and determine the entire best path.

Having temporarily eliminated the *Delete* states, in the following $s \in \{Match, Insert\}$. For $t = 1$, we initialize the highest probability paths to each node and state to simply the probability of being in that node and state and seeing the first observation. Having constructed $\pi'[n_s]$ to represent the probability of starting the observation sequence at n_s, we combine it with the probability of seeing observation $o[1]$ at that node and state:

$$best\_prob[1][n_s] = \pi'[s]B[n_s][o[1]].$$

For subsequent times,

$$best\_prob[t][n_s] = B[n_s][o[t]] \times \max_{\substack{n' \text{ in } [start_{t-1}, end_{t-1}] \\ \text{and } s'}} (best\_prob[t-1][n'_{s'}]A'[n'_{s'}][n_s])$$

and

$$path[t][n_s] = n'_{s'} \text{ maximizing the above.}$$

Thus, $path[t][n_s]$ is the previous step in the highest probability path to state n_s at time t.

When the above quantities have been computed for each t to seq_len, we incorporate the probability of transition to the *End* state from each possible node which may have generated the final symbol. We cannot justify termination of the process in the middle of the model and hypothesis of an *End* state allows us to explain if a sequence's final symbol was not generated by the final node of the model.

Given $best\_prob[seq\_len][n_s]$ for $n \in [start_{seq\_len}, end_{seq\_len}]$ with these slightly

modified values and $paths[seq\_len][n_s]$, we can now determine the single best path through the model for the given sequence. We must work backwards through the sequence, so initially

$$high\_prob(o) = \max_{n'_{s'}}(best\_prob[seq\_len][n'_{s'}])$$

$$best\_path[seq\_len] = n'_{s'} \text{ maximizing the above}$$

and, for each previous step,

$$best\_path[t] = paths[t+1][best\_path[t+1]].$$

Calculation of best path for multi-domain sequences

When we initialize the *start* and *end* arrays, we do so only for a single domain. In the above calculations we generalize these arrays to refer to the appropriate range of nodes on the concatenated models. In particular, to generalize for a multi-domain model we set

$$start_t = (t - \frac{M}{2}) \bmod T$$

and

$$end_t = (t + \frac{M}{2}) \bmod T$$

to refer to the appropriate position on the current model.

2.4.4 Computation time

Evaluating the best path for a given observation sequence requires calculation, for each time t in $[1, seq\_len]$, for each node n in the window $[start_t, end_t]$ and for each symbol producing state s in the node, the probability of transition from each node n' and symbol producing state s'. Because we have incorporated the *Delete* states into the transition probability matrix, n' is not necessarily n or $n-1$ and we must consider the entire window $[start_{t-1}, end_{t-1}]$. Thus, calculation of the best path requires on the

order of $seq\_len \times (M \times N)^2$ operations. Using the left-to-right property of the model, we can eliminate some transitions from consideration, but this does not fundamentally affect the order of operations. Here, we only consider two states: $Match$ and $Insert$ ($N = 2$), but the alignment window, M, is almost on the order of seq_len, which makes this a fairly sizeable computation.

2.4.5 Use of best probability path

We have chosen the probability of the best path through the model as a measure of the model's performance on a sequence, although other measures of distance between a sequence and the model exist. In addition to providing a measure of model performance, the highest probability path guides Baldi *et al.*'s model update procedure.

2.5 Update process

During the model training process, we attempt to maximize the model's performance on the training set (which could equivalently be viewed as the training set's performance on the model). We primarily used the conventional Baum-Welch update procedure described in Rabiner [24]. In addition, we implemented the much less time-consuming update procedure described by Baldi *et al.* in [1, 2].

2.5.1 The Baum-Welch update routine

During this update procedure, we compute intermediate quantities α, β, γ, and ξ for each sample sequence. These facilitate the update process and add a certain amount of intuitiveness to it.

The quantity $\alpha_t[n_s]$ represents the conditional probability of being at (global) state n_s at time t, given the partial observation sequence $o[1, t]$. $\beta_t[n_s]$ represents the conditional probability of being at n_s at time t given the partial observation sequence $o[t+1, seq\_len]$ and $\gamma_t[n_s]$ represents the probability of being at n_s at that time given the entire observation sequence. $\xi_t[n_s]$ is the probability of being in state n_s at time

t and $n'_{s'}$ at time $t+1$. We first use α and β to compute γ and ξ, then use γ and ξ to compute the new parameters of the model.

Scaling

Since the update quantities α and β are extremely small, we compute them in logarithmic scale. Generally, this is very convenient, as we perform many multiplications with these quantities. However, at times in the update procedure we must add a set of quantities stored in log scale. In each case, these quantities represent probabilities and, thus, we can safely assume that their logarithms will take values less than zero. As a convention, when storing quantity x in log scale, if $x = 0$, we let $\log\_x = 1$, a value not possible to achieve for $x \leq 1$, the constraint all x satisfy when x represents a probability.

When called on to add the set of quantities $\{x_n\}$, if we have stored each x_n as $\log\_x_n$, we first compute the average value of $\log\_x_n$ and assign its absolute value to *scale*. We then compute

$$\log(sum) = \log(\sum_{n\ s.t\ \log\_x_n \neq 1} \exp(\log\_x_n + scale)) - scale.$$

We must scale each term to avoid underflow in evaluation of the exponential for each term. However, when the argument of exp() is greater than E (approximately 1×10^{70}), we have overflow. To balance between these two extremes, we attempt to choose a scale which brings as many $\log\_x_n$'s as possible out of the underflow region without overflowing for the largest $\log\_x_n \neq 1$. Therefore, if $\max(\log\_x_n) + scale > E$ for *scale* as above, then we set $scale = E - |\max(\log\_x_n)|$. By limiting *scale* in this way, we may lose the contributions to the above sum of smaller values of x_n. However, if the relative sizes of the largest and smallest logarithms vary so greatly, the smaller values would have had a virtually insignificant effect on *sum* anyway.

Floating point errors not related to overflow or underflow may also cause difficulties in calculating the above sum. For x_n close to one, $\log(x_n)$ may return zero for single precision x_n and the above sum may exceed one inappropriately. Double preci-

sion x_n, combined with restrictions on minimum size for model parameters, appears to circumvent these types of errors.

Updates for multi-domain sequences

For multi-domain sequences, we apply the update routine to overlapping segments of the sequence. We consider segments of length *max_dom* and overlap segments by the window size M. In particular, we break the sequence into *n_dom* domains where

$$n\_dom = \frac{seq\_len}{min\_dom},$$

but we increment *n_dom* by one if $seq\_len - n\_dom \times T \geqq \frac{M}{2}$, i.e., if the end of the sequence exceeds the end of the final model by more than half a window. If we did not increment the number of domains in this case, we would limit the contribution of the final amino acids to the update procedure since computation is performed only within a window about position in the sequence.

In breaking the sequence into n_dom domains, for the d^{th} domain we use the partial observation sequence

$$o^d = o[(d-1) \times min\_dom + 1, d \times min\_dom + M].$$

This allows us to consider the possibility that the amino acids in the range $[d \times min\_dom, d \times min\_dom + M]$ were generated by the d^{th} model or, because of the overlap, by the $(d+1)^{st}$ model. Note that the amino acid $o[d \times max\_dom]$ could not generally have been generated by the d^{th} concatenated model, but to allow for both $o[d \times min\_dom]$ and $o[d \times max\_dom]$ to have been generated by the same model would require excessive overlap for multi-domain sequences. However, the results with this approach to sequence division indicate that an alternative division into segments might prove more appropriate.

When considering multi-domain sequences, we construct an alternative to the *end*

array. For each t, we let

$$end'_t = \min(T+1, t+\frac{M}{2})$$

if we have a multi-domain sequence and are not considering the final domain and

$$end'_t = end_t$$

otherwise.

Calculation of α_t ("forward variables")

For a given observation sequence, $\alpha_t[n_s]$ represents the conditional probability of the partial observation sequence $o[1,t]$, given the model parameters and presence at n_s at time t. Once initialized, we compute these quantities by feeding information forward in time and forward through the model.

The initial probability of being in node one at time one, given $o[1]$, is given for all s by

$$\alpha_1[1_s] = B[1_s][obs]\ \pi[s]$$

where obs = "*skip*" if $s = Delete$ and $o[1]$ otherwise.

However, we must also fill in values for all other n in $[start_1, end_1]$, which we do by acknowledging that passing through the *Delete* state does not increase the time index. Thus, for n in $[start_1 + 1, end'_1]$ and all s,

$$\alpha_1[n_s] = B[n_s][obs]\ \alpha_1[(n-1)_{Delete}]\ A[(n-1)_{Delete}][n_s]$$

where, again, obs = "*skip*" if $s = Delete$ and $o[1]$ otherwise.

For time $t \in [2, seq\_len]$, we use the probabilities calculated for the previous time step. Since we know that $\alpha_{t-1}[n'_{s'}]$ represents the probability of being at $n'_{s'}$ at time $t-1$ given $o[1, t-1]$, we can calculate $\alpha_t[n_s]$ by considering all possible previous positions $n'_{s'}$ in the model and the probability of transition from $n'_{s'}$ to n_s, then

including the probability of seeing observation $o[t]$ at n_s. Since all transitions in the model move at most one node, we need only sum over possible previous states, which will cover all possible previous nodes. Note, however, that if the previous state is *Delete*, we must use probabilities from the current time step because transitions from *Delete* states do not leave the time step. Therefore,

$$\alpha_t[n_s] = B[n_s][obs] \sum_{\substack{s' \text{ s.t.} \\ prev\_n \in [start_{prev\_t}, end_{prev\_t}]}} (\alpha_{prev\_t}[prev\_n_{s'}]\ A[prev\_n_{s'}][n_s])$$

where $prev\_t = t$ if $s' = Delete$ and $t-1$ otherwise, $prev\_n = n$ if $s = Insert$ and $s' \neq Delete$, $n-1$ otherwise, and $obs =$ "$skip$" if $s = Delete$ and $o[t]$ otherwise.

We calculate α_t for nodes within a window and assume that the probability of being at a node outside the window is zero. In addition, our definition of time step requires that the probability of being in a non-symbol producing state (i.e. *Delete*) at the last node in a window for time t be zero. Thus we require $\alpha_t[(end_t)_{Delete}] = 0$. We make an exception for the last node of the model when updating for any but the last segment of a multi-domain sequence. In that case, the alignment window can overlap multiple models and we are only considering the portion of the window on the current model. Thus, even though no symbol could be emitted for time step t on this model, one could be emitted on the next model, so no logical contradiction occurs in letting the model pass through state $\alpha_t[T_{Delete}]$.

Calculation of β_t ("backward" variables)

Because β_t depends on the partial observation sequence $o[t+1, seq\_len]$, we initialize it at the end of a sequence. We set

$$\beta_{seq\_len}[n_s] = 1$$

for all nodes n in the window $start_{seq\_len}$ to end'_{seq_len} and all s. Note that $\alpha_{seq\_len}[n_s]$ represents the probability of being at n_s given the entire observation sequence; initializing $\beta_{seq\_len}[n_s]$ as above allows us to say that $\alpha_{seq\_len}[n_s]\ \beta_{seq\_len}[n_s]$ is the probability

of being at n_s given the entire observation sequence.

For earlier times, we consider all possible subsequent positions, combined with the probability of seeing a particular symbol at that position and the probability of transition from the current position to that position. Note that, analogously to the calculation of α_t, transitions from *Delete* states must not involve an increase in time index. Therefore, in the following $next\_t = t$ if $s = Delete$. Note that these calculations require working backwards through the nodes in the window $[start_t, end'_t]$ as well as through time. Using the convention that $B[n_{Delete}][m] = 1$ for $m =$ "*skip*" (the symbol generated by the *Delete* state), we let

$$\beta_t[n_s] = \sum_{\substack{s' \text{ s.t.} \\ next\_n \in [start_{next\_t}, end_{next\_t}]}} (B[next\_n_{s'}][obs]\ A[n_s][next\_n_{s'}]\ \beta_{next\_t}[next\_n_{s'}])$$

where $next\_n$ is either n or $n+1$, depending on s and s' and $obs =$ "*skip*" if $s' = Delete$ and $o[next\_t]$ otherwise. If $next\_n \notin [start_{next\_t}, end'_{next\_t}]$, we omit the term.

Calculation of γ_t

We calculate α and β to facilitate computation of γ and ξ, which we then use to calculate the new model parameters. Given $\alpha_t[n_s]$ (the probability of being at n_s given the model and the partial observation sequence $o[1,t]$) and $\beta_t[n_s]$ (the probability of being at n_s given the partial observation sequence $o[t+1, seq\_len]$), $\gamma_t[n_s]$ (the probability of being in state n_s given the model and the entire observation sequence) is simple to calculate: it is simply the product of $\alpha_t[n_s]$ and $\beta_t[n_s]$. We normalize to give a probability distribution over possible n_s for time t, because we assume that the sequence must pass through some n_s for n in the window $start_t$ to end'_t at time t.

$$\gamma_t[n_s] = \frac{\alpha_t[n_s]\ \beta_t[n_s]}{\sum_{n'=start_t}^{end'_t} \sum_{s'} (\alpha_t[n'_{s'}]\ \beta_t[n'_{s'}])}$$

which insures that

$$\sum_{n=start_t}^{end'_t} \sum_{s} \gamma_t[n_s] = 1.$$

Note that since we calculate α and β from different ends of the sequence, we must save at least one of these quantities for all t, while we need only save the other long enough to calculate γ_t and ξ_t. We have opted to save the forward variables and, for each time step t, save only β_t and β_{t+1}.

Calculation of ξ_t

ξ_t denotes the probability of being in state n_s at time t and $next\_n_{s'}$ at time $next\_t$ ($t+1$ if $s \neq Delete$ and t otherwise), so summing over $next\_n_{s'}$ gives $\gamma_t[n_s]$. the probability of being in state n_s at time t. Specifically,

$$norm = \sum_{n'=start_t}^{end'_t} \sum_{s_1,s_2} \alpha_t[n'_{s_1}]\ A[n'_{s_1}][next\_n'_{s_2}]\ B[next\_n'_{s_2}][o[t]]\ \beta_{next\_t}[next\_n'_{s_2}]$$

$$\xi_t[n_s][next\_n_{s'}] = \frac{\alpha_t[n_s]\ A[n_s][next\_n_{s'}]\ B[next\_n_{s'}][o[t]]\ \beta_{next\_t}[next\_n_{s'}]}{norm}.$$

We only compute ξ_t for $t = 1, seq\_len - 1$ because $next\_t$ will extend past the end of the sequence for all states but *Delete* . However, if we have a single domain sequence or the last segment of a multi-domain sequence, we update at $t = seq\_len$ for the transition to the *End* state by letting

$$\xi_{seq\_len}[n_s][End] = \gamma_{seq\_len}[n_s]$$

for symbol-producing states s.

Use of the update quantities

The above quantities are computed for each sequence or segment of a sequence in the training set and we use them to calculate the contribution to the new model parameters.

The following update rules stem from an interpretation of computing probability by counting event occurrences. In particular, we can consider $\gamma_t[n_s]$, which is the probability of being in n_s at time t, to be the expected number of times the process is

in state n_s at time t. Similarly, $\xi_t[n_s][next\_n_{s'}]$ can be viewed as the expected number of transitions from n_s to $next\_n_{s'}$. In the following expressions, s is any state, with the exception of the expression for $\overline{B}[n_s][m]$, for which $s \in \{Match, Insert\}$.

$$\overline{\pi}[s] = \frac{\sum_{i=1}^{n\_seg}(\gamma_1[1_s])_i}{n\_seg} = \text{expected number of times in state } s \text{ of node 1 at time 1}$$

$$\begin{aligned}\overline{B}[n_s][m] &= \frac{\sum_{i=1}^{n\_seg}(\sum_{\substack{t=start_n \\ s.t.\ o[t]=m}}^{end_n}\gamma_t[n_s])_i}{\sum_{i=1}^{n\_seg}(\sum_{t=start_n}^{end_n}\gamma_t[n_s])_i} \\ &= \frac{\text{expected number of times in } n_s, \text{ seeing symbol } m}{\text{expected number of times in } n_s}\end{aligned}$$

$$\begin{aligned}\overline{A}[n_s][next\_n_{s'}] &= \frac{\sum_{i=1}^{n\_seg}(\sum_{t=start_n}^{\min(T-1,end_n)}\xi_t[n_s][next\_n_{s'}])_i}{\sum_{i=1}^{n\_seg}(\sum_{t=start_n}^{end'_n}\gamma_t[n_s])_i} \\ &= \frac{\text{expected number of transitions from } n_s \text{ to } next\_n_{s'}}{\text{expected number of transitions from } n_s}.\end{aligned}$$

Note that we do not update for transitions from node T, although we do use multi-domain sequences. For the majority of the sequences/segments used, $\xi_t[T_s][next\_n_{s'}]$ is minuscule except for transitions to $Insert$ and averaging over all segments causes inappropriately small transitions at the last node of the model.

After applying these update rules, we must sometimes adjust the model parameters in a less elegant manner. Floating point errors can prevent quantities that should sum to exactly one from doing so, so we make adjustments to compensate for those errors. In addition, if a particular transition rarely or never occurs, or a certain symbol never appears at a position within a training set, the update rules will cause the corresponding parameters to go to zero. However, if we allow any model parameters to go to zero, we introduce the possibility that a sequence which closely matches the model for the most part may have zero probability of having been generated by the model. To avoid this situation, we set any quantities that have been updated to an

excessively small value to a small but non-zero constant.

Training time

When calculating the α's, we consider for each time step all the nodes n and states s in which the process could possibly be at time t and all n', s' where it could have been at time $t-1$ (or at time t if $s = Delete$). However, given n, s, and s' we can determine n', so calculation of α for a sequence only requires time on the order of $seq\_len \times M \times N^2$. Similarly, calculation of β and ξ require the same approximate number of operations. Calculation of γ requires only $seq\_len \times M \times N$ operations because its calculation does not require consideration of transitions between time steps. When calculating these quantities, we use segments of size $seq\_len \approx T$, so we can say that the calculation of γ and ξ for each sequence requires time on the order of $T \times M \times N^2$. Given these quantities for each sequence, calculation of the new model parameters requires that we calculate for each node in the model the new probability of transitions from all states s to all states s' and, for each state s, the probability of seeing each symbol m. This implies operations at least on the order of $T \times N \times (N + S)$, regardless of update method, but that quantity is dominated in this case by $n\_seq \times T \times M \times N^2$ since $N = 3$, $S \leq 20$ and $M \approx 60$.

2.5.2 The Baldi update

One appeal of this alternative to the Baum-Welch update routine is that updates of model parameters cannot cause quantities to become zero. The re-parameterization of the model parameters guarantees this, as well as guaranteeing that the appropriate quantities will sum to one. However, due to floating point errors, we do in fact set a minimum value for model parameters and adjust the probabilities if necessary.

Model parameterization

We define $w[n_s][n'_{s'}]$ and $v[n_s][k]$ such that

$$A[n_s][n'_{s'}] = \frac{\exp(w[n_s][n'_{s'}])}{\sum_{s''} \exp(w[n_s][n'_{s''}])}$$

and

$$B[n_s][k] = \frac{\exp(v[n_s][k])}{\sum_{k'=1}^{S} \exp(v[n_s][k'])}.$$

Although the transition probability matrix used by Baldi *et al.* very likely contains the information contained in what we call π, we explicitly define $x[s]$ such that

$$\pi[s] = \frac{\exp(x[s])}{\sum_{s'} \exp(x[s'])}.$$

Update procedure

This update procedure does not require calculation of γ or ξ, which saves a great deal of computing time. In fact, the update relies solely on the best path through the model, which we compute at each step anyway as a means of evaluating model performance. The approach was designed to be used with updates after the presentation of each sample sequence, but we have chosen to use it in batch mode with updates performed after the presentation of all sample sequences. For each sample, we compute the following contribution to $\Delta x[s]$:

$$\Delta x_i[s] = \eta(X_s - \pi[s])$$

where X_s is one if the best path passes through node 1, state s at time 1 and zero otherwise.

When updating for state transitions and symbol emissions, we must consider the implicit *Delete* states in the best path. If the node associated with time t in the best path ($best\_node_t$) is more than a single node away from $best\_node_{t-1}$, then we must have passed through one or more *Delete* states at time t before reaching $best\_node_t$. Therefore, when using this update routine we incorporate the *Delete* states back into the best path.

At each time t we compute a contribution to $\Delta w[n_s][next\_n_{s'}]$ for all s and all $n \in [best\_node_{t-1}, best\_node_t)$. In this way, we address the transitions from $best\_node_{t-1}$, transitions to $best\_node_t$, and all transitions between. To update for the symbol emission probability matrix, we compute a contribution to $\Delta v[n_s][m]$ for all s and each $n \in (best\_node_{t-1}, best\_node_t]$. This slightly different range of nodes corresponds to associating a sequence of *Delete*'s, followed by $o[t]$, with time t. If $best\_node_{t-1} = best\_node_t$, we look only at node n.

The contribution of sample i to the nodes in the above range is given by

$$\Delta w[n_s][next\_n_{s'}] = \eta(T[n_s][next\_n_{s'}] - A[n_s][next\_n_{s'}])$$

$$\Delta v_i[n_s][m] = \eta(E[n_s][m] - B[n_s][m])$$

and where $T[n_s][next\_n_{s'}]$ is one if the transition n_s to $next\_n_{s'}$ occurs in the best path for sequence o_i and zero otherwise. Similarly, $E[n_s][m]$ is one if $o_i[t] = m$ and $best\_path_t = n_s$ and zero otherwise. For transitions to the *End* state, we let $T[n_s][End] = 1$ if and only if $best\_path_{seq\_len} = n_s$ and set it to zero for all other n_s.

After computing the contributions to the Δ's from each sequence we compute an average Δ by dividing by the number of sequences considered. Then the new model parameters are given by

$$\overline{\pi}[s] = \frac{\exp(x[s] + \Delta x[s])}{\sum_{s'} \exp(x[s'] + \Delta x[s'])}$$

$$\overline{A}[n_s][next\_n_{s'}] = \frac{\exp(w[n_s][next\_n_{s'}] + \Delta w[n_s][next\_n_{s'}])}{\sum_{s''} \exp(w[n_s][next\_n_{s''}] + \Delta w[n_s][next\_n_{s''}])}$$

and

$$\overline{B}[n_s][m] = \frac{\exp(v[n_s][m] + \Delta v[n_s][m])}{\sum_{m'} \exp(v[n_s][m'] + \Delta v[n_s][m'])}$$

and w, v, and x are all incremented by the corresponding Δ for the next iteration.

Training time

Since we do not need to consider alignments in calculating the contribution of each sequence to the new model parameters, the time required to calculate these contributions is on the order of $n\_seq \times T \times N \times (N + S)$, or roughly a factor of M less than the Baum-Welch update routine.

2.5.3 Convergence

Convergence in the training process is extremely difficult to define with any great certainty. One approach is simply to train a model until the increase in average probability is "negligible." However, the numerical scale of the probabilities is so small and varies so much across data sets that "negligible" improvement is a rather ambiguous term. We could, alternately, update each model for a fixed number of iterations, but that assumes that all models require a comparable amount of training time. In fact, the training sets do not appear to attain a comparable average probability on the different models in the same number of iterations. Another possibility is to require improvement on at least a certain fraction of the training samples. Since the update process is averaged over training samples, there may not be improvement on 100% of training samples at each iteration. If the number of samples showing improvement dwindles, it may mean that we are adjusting the model to emphasize a few high probability sequences at the expense of a larger number of lower probability sequences. However, this criterion cannot be applied strictly because in many cases the percentage of samples showing improvement dwindles, only to rise again sharply after a few additional iterations. In addition, this criterion may be met while the average probability still shows "negligible" improvement, as defined by virtually no percent change from one iteration to the next.

Given the benefits and failings of each of the above criteria, we use a combination in determining when to stop training and which iteration to call the "best." In particular, we track the number of iterations with relative and absolute decrease in average and average log of probability. We use the average log of probability because

it provides a somewhat more robust measure of training set performance, given the possibility of underflow when computing average probability directly.

If we train for *n_rel* (set to five) iterations with a relative decrease in performance on both of these measures, we assert that the model is diverging and cease to train. Alternately, we may see a single relative decrease from one iteration to the next, then a series of relative increases without ever performing better than the initial iteration. In this case, we have a series of iterations with absolute decrease in performance without a series of relative decreases. This situation offers more hope, as it does not indicate divergence. Therefore, we allow up to *n_abs* (set to ten) iterations with an absolute decrease in probability as long as the relative performance continues to improve. In general, a relative increase in probability will either produce the best probability to date or exceed it within a few steps.

We define "negligible" improvement in relative (not absolute) performance as improvement of less than *neg_pct* percent (set to ten) in average probability and improvement of less than neg_fract (set to .1) in average log probability. If we have negligible improvement by this definition, or if we have not seen improvement on at least *best_pct* (65) percent of the samples for *n_neg* (set to ten) iterations, we assert that the model has converged and stop training. This last criterion must be applied with some discretion. In some cases, the model has shown an initial drop in average probability, followed by a steady rise, but without improvement on *best_pct* within *n_neg* iterations. In these cases, we begin training again from the point at which it stopped.

After one of the above criteria has caused the model to stop training, we take and save the last iteration which showed improvement on *best_pct* percent or greater of the training samples.

Chapter 3 Modeling with real vector representations of sequences

3.1 Preliminary processing of protein sequences

Considering protein folding as a signal recognition problem requires conversion from the sequence of symbols representing amino acids to a vector of real numbers. There are many options in this process, depending on what characteristics of the sequence we seek to represent.

3.1.1 Property profiles

One option for real vector representation of a protein sequence is to choose a biochemical property and convert the amino acid sequence into a "property profile." To generate a property profile, we replace each amino acid in a sequence by its value in a property table. Note that while these profiles represent the sequence with respect to the chosen property to some extent, the representation is hardly definitive.

Property tables have been calculated by many different groups [10, 11] and even if these tables represent the same property, they may vary significantly. In addition, some properties have high theoretical correlations to others, but, given the variations between tables for the same property, there seems no harm in considering correlated properties as the correlation may not even appear in the table.

We have considered the following "six" properties: hydrophobicity (PONNU), charge, accessible surface area, van der Waals volume, hydrophobicity (ROSEF), and hydrophobicity (PRIFT) [10]. Given this set of six property tables, we normalized each to zero mean and unit variance across the amino acids to reduce magnitude variations.

After generating the property profiles, we could pass them directly into the model and proceed completely analogously to the discrete symbol case. However, we choose to perform some additional processing. The unprocessed property profile model requires an excessive amount of computation and cannot be expected to behave much differently than the symbolic case, with twenty different floating point numbers representing the symbols instead of twenty integers.

To process the profiles, we treat the property profile vector as a signal from the hypothetical random process and compute the Fourier power spectrum. To facilitate the use of the Fast Fourier Transform, we select sequences from a specified range and pad them to a length which is a power of two with zeros. Given the training set of transformed profiles, we compute the average power of each component and use that as a measure of the "significance" of the components. We retain the n_sig highest power components of the training sequences. The components retained then become a fixed property of the model: we use them to represent all sequences passed to the model and all non-training sequences are reduced, after the FFT, to the indicated components.

We consider processed property profiles partially because of sequence length variation within structural classes: transforming the profile allows us to capture global sequence information within a single component. In addition, we can reduce the dimension of the problem somewhat by retention of components we believe to be significant in some way. We use the Fourier transform and retain high power components, although other transform-related options certainly exist.

3.1.2 Amino acid composition profiles

Another property of amino acid sequences we can consider is the amino acid composition. Chou and Zhang [7, 8, 9] achieved very good results classifying sequences from the PDB into the classes α, β, $\alpha + \beta$ and α/β, so we have considered this representation for our continuous symbol model.

To convert the amino acid sequence into this format, we compute, over the length

of the sequence, the average number of times a given amino acid appears. In this way, we reduce the sequence to a 19-dimensional vector whose n^{th} entry is the percentage of times the n^{th} amino acid appears in the sequence. The 20^{th} component is completely determined by the others and is therefore omitted. This crudely constructed quantity appears to have some merit in distinguishing between some structural classes, although perhaps not as much as indicated by the results of Chou and Zhang.

3.2 Model adaptation

To handle observations drawn from a continuous distribution, we must make some alterations to the model, primarily related to the observation probability at each node.

3.2.1 Non-discrete observation distributions

If the observation sequence o does not take on a discrete set of values, we must replace the symbol emission probability matrix B with a probability density function. The alphabet S then becomes a set of mixtures from which a given observation may have been drawn. B is characterized by the statistics of these mixtures. Thus,

$$B[n_s](o[t]) = \sum_{m=1}^{S} (c[n_s][m]\ \mathfrak{N}(o[t], \mu[n_s][m], \sigma[n_s][m]))$$

where c is the mixture coefficient, μ is the mixture mean and σ is the mixture variance and $\mathfrak{N}$ is a Gaussian density function. B no longer represents a matrix, but rather an array of functions.

To calculate $B[n_s](o[t])$ for use in the Baum-Welch update procedure, we use the Gaussian probability density function with the appropriate mean and variance. This is not strictly a probability, but any scaling factor would cancel out in the update routine and multiplication by a sliver ε would incorporate a constant factor into the best probability path for a sequence. Just as we prevent model quantities in the discrete symbol case from becoming too small, we prevent the variance of each mixture

from becoming too small. In addition, this allows us to insure that the probability density function will not return a value exceeding one.

Note that it is not strictly necessary, given the above ε-scaling argument, for the B's to return values not exceeding one. However, the current log scale storage of small quantities would be complicated by the new possibility of achieving the value one. Alternative scalings exist, but our restrictions on size of variance have not seemed to harm significantly the performance of the model.

3.2.2 Initialization of continuous observation distributions

For observations drawn from continuous distributions, we initialize the mixture statistics by calculating the sample mean and variance for each position in the test set sequences. Initially we set the mixture coefficients (c) to uniform probability over mixtures and the mixture means to the sample means. The variances are set to values about the actual sample variance on the training set.

$$c[n_s][m] = \frac{1}{S}$$

$$\mu[n_s][m] = \frac{1}{n\_seq} \sum_{i=1}^{n\_seq} o_i[n]$$

and

$$v[n_s][m] = \left(\frac{1}{n\_seq} \sum_{i=1}^{n\_seq} (o_i[t] - \mu[n_s][m])^2\right) \pm \Delta_m$$

for each mixture m and each state s.

We select Δ's such that $\Delta_1 = 0$ and we set the first mixture's variance to the sample variance. For mixtures 2 to S, we attempt to let the variances range from approximately half the sample variance to approximately 1.5 times the sample variance.

3.2.3 Alignment modifications

Replacement of the B matrix by a function is the only required difference in model architecture between these two types of observation sequences. However, we have made a few additional adjustments. For the transformed profile data and amino acid composition profiles, we do not consider alignment within a window. Having replaced amino acid position with Fourier power spectrum components (or with average amino acid composition), position in the sequence corresponds to a specific component and it becomes inappropriate to consider aligning sequences except to the known corresponding node. Thus, model node and time step become the same quantity. Initially, we set $\pi[Match] = 1$ and $A[n_{Match}][next\_n_{Match}] = 1$ for each n, which insures that we never deviate from the main line of the model. And, after updates, we no longer adjust the probabilities to insure that no quantities achieve the value zero as we did with the discrete symbol case.

3.2.4 Modification of the Baum-Welch update routine

Calculation of updates to the state probability and transition arrays remains the same as for the discrete symbol case. However, we must now calculate updates for the mixture statistics at each node and state.

Evaluation of γ_t

The update variables α, β and ξ remain the same, but γ is slightly modified. In the continuous observation distribution case, $\gamma_t[n_s][m]$ now represents the probability of being at n_s and in mixture m at time t, with the sum over mixtures giving us, again, the probability of being at n_s at time t. However, this quantity is trivial now that we have removed alignment considerations because $end_t = start_t = t$. Note also that in each of the following equations $n = t$ and $s = Match$.

$$\gamma_t[n_s][m] = \frac{c[n_s][m]\ \mathfrak{N}(o[t], \mu[n][m], \sigma[n][m]}{\sum_{m'} \mathfrak{N}(o[t], \mu[n_s][m'], \sigma[n_s][m']} \times \frac{\alpha_t[n_s]\ \beta_t[n_s]}{\sum_{n'=start_t}^{end'_t} \sum_{s'} (\alpha_t[n'_{s'}]\ \beta_t[n'_{s'}])}$$

$$= \frac{c[n_s][m]\ \mathfrak{N}(o[t], \mu[n][m], \sigma[n][m]}{\sum_{m'} \mathfrak{N}(o[t], \mu[n_s][m'], \sigma[n_s][m']}$$

Update of mixture statistics

To update the mixture coefficients, we consider the number of times in n_s, mixture m, over the number of times in n_s:

$$\begin{aligned} c[n_s][m] &= \frac{\sum_{i=1}^{n\_seq} \sum_{t=start_n}^{end_n} (\gamma_t[n_s][m])_i}{\sum_{i=1}^{n\_seq} \sum_{t=start_n}^{end_n} \sum_{m'} (\gamma_t[n_s][m'])_i} \\ &= \frac{\sum_{i=1}^{n\_seq} (\gamma_t[n_s][m])_i}{\sum_{i=1}^{n\_seq} \sum_{m'} (\gamma_t[n_s][m'])_i}. \end{aligned}$$

To compute the new mixture average, we use

$$\mu[n_s][m] = \frac{\sum_{i=1}^{n\_seq} (\gamma_t[n_s][m])_i\ o_i[t]}{\sum_{i=1}^{n\_seq} \sum_{m'} (\gamma_t[n_s][m'])_i}$$

and, similarly,

$$v[n_s][m] = \frac{\sum_{i=1}^{n\_seq} ((\gamma_t[n_s][m])_i\ (o_i[t] - \mu[n_s][m])^2)}{\sum_{i=1}^{n\_seq} \sum_{m'} (\gamma_t[n_s][m'])_i}.$$

3.2.5 Computation time

The time required for training a model with our real vector representations is significantly less than that for symbolic representations because of the lack of alignment considerations. In particular, calculation of the Baum-Welch update quantities now requires only $T \times N^2$ operations for α, β and ξ. The new γ requires calculations for each mixture, which introduces an extra factor of S into the calculation of γ. However, we do not even need to consider the transitions between states because we adhere strictly to the main line of the model, which reduces the training time to approximately order $T \times S$ for each sequence.

Chapter 4 Experiments

Given the symbolic and floating point HMMs, and the multiple representations within these types of models, how do these various constructions perform as classifiers? We trained a large number of models for the Globin, Calcium-Binding, Kinase and Random (or "other") classes and measured their effectiveness at discriminating between these classes. For the symbolic models, we considered various groupings of amino acids to reduce the alphabet size. Since we already know that HMMs are effective at discriminating between structural classes of proteins, in this way we hoped to gain insight into the biochemical forces driving the folding process and the redundancy of the protein alphabet. We also seek this insight with the real vector representations. In addition, we hoped to ascertain the effectiveness of the HMM approach on floating point representations of protein sequences.

The sequences for these experiments were drawn from the SWISS-PROT database. The Calcium-Binding, Globin, and Kinase class information was taken from the file keywords.txt accompanying the database. The "Random" class consists of sequences from the database with lengths in the range [75,250] drawn from twelve other classes listed in the keywords.txt file, with at most 50 sequences drawn from a single class.

This is only one possible method of construction of "Random" classes. We chose this construction primarily to insure a rigorous test of model performance. Randomly generated sequences may have very little resemblance to real protein sequences. In particular, a random sequence of amino acids may not fold to a compact shape the way a real protein would. If we used randomly generated sequences to train our Random model, we might not be testing the ability of our models to detect proteins from the classes they are trained for, but rather their ability to discriminate between proteins and non-proteins. By using real protein sequences, we provide a realistic test of the more difficult procedure of differentiating between classes.

4.1 Symbolic alphabets

In addition to the full twenty letter amino acid alphabet, we considered several other groupings of the amino acids based on various criteria. For each of these groupings, the fundamental data remained constant. We trained models for the Globin, Calcium-Binding (CB), Kinase and Random classes for each alphabet and evaluated their performance on the data sets for each of these classes. This allowed construction of a maximum-likelihood classifier in which a sample was assigned to the class on whose model it attained the highest probability.

4.1.1 Data sets

Since we considered the Globin and CB sequences as potentially multi-domain, we could use most of the sequences in the database, discarding only those with length less than the chosen minimum domain length. The fact that we could process sequences of any length as multi-domain sequences gave us an adequately sized data set for the CB class. Note that any sequence passed to the model for a multi-domain class was treated as multi-domain. We used 669 Globin sequences in total, randomly selecting approximately 70% of them (468) as a training set and retaining the remainder as a test set. Similarly, we used 423 CB sequences with a training set of size 296. Kinase is a single domain class, but the database contains sequences over a thousand amino acids in length. For computational practicality, we restricted our data set to those sequences falling in the length range [256,512]. This gave us a total set size of 245 sequences with a training set of size 171. The Random class contains a total of 356 sequences, with a training set of size 249. These appear to be adequately sized training sets to allow generalization to the test sets, as the two sets performed with approximately the same accuracy on most of the alphabets.

The "other" symbol

For each of the following symbolic alphabets, we included an additional symbol to represent any ambiguous information from the database. For example, in some se-

quences, the symbol "X" appears to indicate that the amino acid in that position is unknown. While this does not occur frequently, incorporation of the "other" symbol allows us to handle this symbol without discarding the entire sequence.

4.1.2 Variance in classification accuracy

In an attempt to estimate the variance in test set performance over test set composition, we considered divisions of the test set into sub-sets and computed the average classification accuracy for each sub-set. We then computed the variance over this set of sub-sets and averaged that quantity over a large number of such divisions. This procedure indicated a standard deviation on the order of 2.5% for the Globin class for each alphabet considered, 5.5% for the CB class and 4% for the Random class. The Kinase class demonstrated large variations for the full and chemical alphabets (on the order of 8% and 5% respectively), but dropped to something less than 2% for the other reduced alphabets.

4.1.3 The full alphabet

The full twenty letter alphabet undoubtedly gives the best results overall when attempting to differentiate between CB, Globin and Random classes but for some reason does not perform exceptionally well on the Kinase class.

Symbols		Globin		CB		Kinase		Random	
20		%	#	%	#	%	#	%	#
Globin	Train	98.72	462	1.28	6	0.00	0	0.00	0
	Test	97.01	195	2.49	5	0.00	0	0.50	1
	Total	**98.21**	**657**	**1.64**	**11**	**0.00**	**0**	**0.15**	**1**
CB	Train	0.00	0	98.99	293	0.00	0	1.01	3
	Test	0.00	0	97.64	124	0.00	0	2.36	3
	Total	**0.00**	**0**	**98.58**	**417**	**0.00**	**0**	**1.42**	**6**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	52.70	39	47.30	35	0.00	0
	Total	**0.00**	**0**	**15.92**	**39**	**84.08**	**206**	**0.00**	**0**
Random	Train	0.00	0	0.80	2	0.00	0	99.20	247
	Test	0.00	0	27.10	29	0.00	0	72.90	78
	Total	**0.00**	**0**	**8.71**	**31**	**0.00**	**0**	**91.29**	**325**

Table 4.1: Sequences assigned to each class by 20-Symbol models

We assign a sequence to a class if the probability of its "best" path through a model is higher than that for other models and, in this representation, the sequences belonging to the training sets almost all score higher on the appropriate classes. Generalization to the test sets is excellent for the Globin and Calcium-Binding classes, fair for the Random class and poor for the Kinase class.

It is unclear what influences caused the relatively poor performances on the test sets of Random and Kinase. We could hypothesize that the training set for Kinase is of inadequate size, since it is more than 120 sequences smaller than the CB training set, but that does not explain how alternate alphabets overcome this deficiency. Similarly, the errors on the test set of the Random class could be a consequence of the fact that, by construction, the Random sequences do not have high structural or sequence similarities. However, with most alternate representations generalization to the Random test set occurs with no difficulty.

Although the CB class performs quite well with this representation of sequences, it is interesting to note that the number of *Match* states in the "best" paths for the CB training sequences is generally much lower than expected. We would expect a number of *Match*'s approximately equal to the estimated number of domains times the model length, but we generally saw a number approximately the (single) model length. This indicates some difficulty with the treatment of multi-domain sequences, although this difficulty does not adversely affect performance in this case.

4.1.4 Reduced alphabets

We considered several alphabet reductions based on the properties of hydrophobicity and polarity. In addition, we considered several other common groupings of amino acids. To varying degrees, almost all of these alphabets demonstrated the ability to differentiate between at least the Globin, Kinase and Random classes. For the Calcium-Binding sequences, the poor performance and occasional failure of the training process most likely had to do with the treatment of multi-domain sequences.

The 8-letter "chemical" alphabet

In this alphabet, we group the amino acids based on a set of chemical properties. The groups are {Asp, Glu} (acidic), {Arg, His, Lys} (basic), {Ala, Gly, Ile, Leu, Val} (aliphatic) {Asn, Gln} (amide), {Phe, Trp, Tyr} (aromatic), {Ser, Thr} (hydroxyl), {Pro} (imino) and {Cys, Met} (sulfur) [27].

While the results with this alphabet are worse with respect to the classification of the CB sequences, performance on Kinase and Random is significantly improved. The CB sequences mis-classified are almost all longer than 200 amino acids, indicating that the difficulty in classification may have to do with the treatment of multi-domain sequences. We see a larger number of *Match* states in the best paths, but this coincides with reduced performance.

Symbols		Globin		CB		Kinase		Random	
8		%	#	%	#	%	#	%	#
Globin	Train	98.50	461	0.00	0	1.07	5	0.43	2
	Test	98.01	197	0.00	0	1.99	4	0.00	0
	Total	**98.36**	**658**	**0.00**	**0**	**1.35**	**9**	**0.30**	**2**
CB	Train	11.15	33	76.69	227	6.42	19	5.74	17
	Test	11.02	14	74.02	94	11.02	14	3.94	5
	Total	**11.11**	**47**	**75.89**	**321**	**7.80**	**33**	**5.20**	**22**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	9.46	7	0.00	0	89.19	66	1.35	1
	Total	**2.86**	**7**	**0.00**	**0**	**96.73**	**237**	**0.41**	**1**
Random	Train	0.00	0	0.00	0	0.00	0	100.00	249
	Test	1.87	2	3.74	4	4.67	5	89.72	96
	Total	**0.56**	**2**	**1.12**	**4**	**1.40**	**5**	**96.91**	**345**

Table 4.2: 8-Symbol "Chemical" alphabet model classifications

The 3-letter "structural" alphabet

In the structural alphabet, amino acids are grouped together according to their preference for position within the tertiary structure. The groups are {(Ala, Cys, Gly, Pro, Ser, Thr, Trp, Tyr} ("ambivalent"), {Arg, Asn, Asp, Gln, Glu, His, Lys} ("external") and {Ile, Leu, Met, Phe, Val} ("internal") [27].

This is a relatively effective grouping of the amino acids for structural classification. Since the grouping derives from structure, we would expect this alphabet to perform well. Performance on the Kinase test set is perfect, although a large number of CB sequences are also classified as Kinase, corresponding to a high false positive rate. The presence of Kinase in this classifier highlights the difficulty in the treatment of the CB multi-domain sequences because, in its absence, models trained with this alphabet perform nearly as well as those trained with the full alphabet.

Symbols		Globin		CB		Kinase		Random	
3 v5		%	#	%	#	%	#	%	#
Globin	Train	95.94	449	1.07	5	1.71	8	1.28	6
	Test	96.02	193	0.50	1	2.49	5	1.00	2
	Total	**95.96**	**642**	**0.90**	**6**	**1.94**	**13**	**1.20**	**8**
CB	Train	0.34	1	72.64	215	17.23	51	9.80	29
	Test	0.79	1	66.14	84	22.83	29	10.24	13
	Total	**0.47**	**2**	**70.69**	**299**	**18.91**	**80**	**9.93**	**42**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	0.00	0	100.00	74	0.00	0
	Total	**0.00**	**0**	**0.00**	**0**	**100.00**	**245**	**0.00**	**0**
Random	Train	0.00	0	0.80	2	0.00	0	99.20	247
	Test	0.00	0	2.80	3	1.87	2	95.33	102
	Total	**0.00**	**0**	**1.40**	**5**	**0.56**	**2**	**98.03**	**349**

Table 4.3: 3-Symbol "structural" alphabet model classifications

Hydrophobic/Polar alphabet reductions

We considered several alphabet reductions based on grouping amino acids together on the properties of hydrophobicity and polarity. At the highest level of differentiation, we consider the groups hydrophobic {Ala, Phe, Ile, Leu, Met, Pro, Val} (H), polar uncharged {Cys, His, Asn, Gln, Ser, Thr, Trp, Tyr} (P_0), polar positively charged {Lys, Arg} (P_+), polar negatively charged {Asp, Glu} (P_-) and Glycine {Gly} (G), giving us a 5-letter alphabet [6] Note that there is a correlation between H and the "internal" group of the "structural" alphabet above–hydrophobic residues tend to prefer the interior of a folded protein.

This alphabet performs essentially as well on the Globin and Random classes as the 8-letter alphabet, but with improved performance on Kinase and decreased performance on CB. Note that in the absence of the Kinase model, models trained with

this alphabet perform nearly as well as those using the full alphabet, differentiating well between the CB, Globin and Random sequences.

Symbols		Globin		CB		Kinase		Random	
5		%	#	%	#	%	#	%	#
Globin	Train	96.15	450	0.00	0	2.14	10	1.71	8
	Test	95.52	192	0.50	1	2.49	5	1.49	3
	Total	**95.96**	**642**	**0.15**	**1**	**2.24**	**15**	**1.64**	**11**
CB	Train	0.00	0	63.51	188	27.70	82	8.78	26
	Test	0.00	0	60.63	77	29.92	38	9.45	12
	Total	**0.00**	**0**	**62.65**	**265**	**28.37**	**120**	**8.98**	**38**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	0.00	0	100.00	74	0.00	0
	Total	**0.00**	**0**	**0.00**	**0**	**100.00**	**245**	**0.00**	**0**
Random	Train	0.00	0	0.40	1	0.40	1	99.20	247
	Test	0.00	0	0.93	1	3.74	4	95.33	102
	Total	**0.00**	**0**	**0.56**	**2**	**1.40**	**5**	**98.03**	**349**

Table 4.4: $H/P_0/P_+/P_-/G$ alphabet model classifications

To reduce the alphabet still further, we incorporate the mildly hydrophobic Glycine into the H group. The performance with this alphabet is comparable to that of the 5-letter alphabet. The absorption of Glycine into the Hydrophobic group appears to have had very little impact on performance, except to slightly improve recognition of CB sequences.

Symbols		Globin		CB		Kinase		Random	
4		%	#	%	#	%	#	%	#
Globin	Train	96.58	452	0.00	0	1.92	9	1.50	7
	Test	94.53	190	0.50	1	1.99	4	2.99	6
	Total	**95.96**	**642**	**0.15**	**1**	**1.94**	**13**	**1.94**	**13**
CB	Train	7.09	21	67.23	199	18.24	54	7.43	22
	Test	4.72	6	66.14	84	20.47	26	8.66	11
	Total	**6.38**	**27**	**66.90**	**283**	**18.91**	**80**	**7.80**	**33**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	0.00	0	100.00	74	0.00	0
	Total	**0.00**	**0**	**0.00**	**0**	**100.00**	**245**	**0.00**	**0**
Random	Train	0.00	0	0.40	1	0.00	0	99.60	248
	Test	0.93	1	2.80	3	4.67	5	91.59	98
	Total	**0.28**	**1**	**1.12**	**4**	**1.40**	**5**	**97.19**	**346**

Table 4.5: Models using 4-letter $H/P_0/P_+/P_-$ alphabet

The next alphabet considered groups the amino acids in a manner very similar to the previous one. This alphabet clusters the amino acids into hydrophobic (H'), acidic (A), basic (B) and uncharged (U) groups. A and B should be identical to P_- and P_+ and H' should obviously be identical to H, but different sources give different groupings of the amino acids. The groupings differ on three amino acids: Glycine is grouped with U instead of being its own group or incorporated into hydrophobic; Histadine is grouped with B instead of (polar) uncharged; and Tryptophan is grouped with H' instead of (polar) uncharged. This alphabet is referred to as the "functional" alphabet.

These three re-assignments of amino acids appear to lead to a slight improvement in correct classification of CB sequences, indicating that at least one of these amino acids has probably been assigned to a more accurate grouping.

Symbols		Globin		CB		Kinase		Random	
4 v1		%	#	%	#	%	#	%	#
Globin	Train	96.15	450	0.21	1	1.71	8	1.92	9
	Test	95.52	192	0.50	1	2.49	5	1.49	3
	Total	**95.96**	**642**	**0.30**	**2**	**1.94**	**13**	**1.79**	**12**
CB	Train	4.73	14	73.99	219	12.16	36	9.12	27
	Test	3.15	4	70.87	90	18.11	23	7.87	10
	Total	**4.26**	**18**	**73.05**	**309**	**13.95**	**59**	**8.75**	**37**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	1.35	1	0.00	0	98.65	73	0.00	0
	Total	**0.41**	**1**	**0.00**	**0**	**99.59**	**244**	**0.00**	**0**
Random	Train	0.00	0	0.00	0	0.00	0	100.00	249
	Test	0.00	0	1.87	2	2.80	3	95.33	102
	Total	**0.00**	**0**	**0.56**	**2**	**0.84**	**3**	**98.60**	**351**

Table 4.6: Models using 4-letter "functional" alphabet

The next reduction involves incorporating all types of polar amino acids into one group, P. We now have a three letter alphabet: $H/P/G$. Without the Kinase model, the performance with this alphabet is comparable to the 5-letter alphabet, which again isolated Glycine. This could indicate that the charge of a polar amino acid is not as significant a force in the folding process as the mere fact that it is polar. However, with the Kinase model, this alphabet performs poorly on CB and with very little change on the other classes.

Symbols		Globin		CB		Kinase		Random	
3		%	#	%	#	%	#	%	#
Globin	Train	94.66	443	0.00	0	2.35	11	2.99	14
	Test	94.53	190	0.50	1	1.99	4	2.99	6
	Total	**94.62**	**633**	**0.15**	**1**	**2.24**	**15**	**2.99**	**20**
CB	Train	0.00	0	56.76	168	30.41	90	12.84	38
	Test	0.00	0	55.12	70	33.07	42	11.81	15
	Total	**0.00**	**0**	**56.26**	**238**	**31.21**	**132**	**12.53**	**53**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	0.00	0	100.00	74	0.00	0
	Total	**0.00**	**0**	**0.00**	**0**	**100.00**	**245**	**0.00**	**0**
Random	Train	0.00	0	0.00	0	0.40	1	99.60	248
	Test	0.00	0	1.87	2	7.48	8	90.65	97
	Total	**0.00**	**0**	**0.56**	**2**	**2.53**	**9**	**96.91**	**345**

Table 4.7: Models using 3-letter $H/P/G$ alphabet

The next alphabet considered is referred to as the "Charge" alphabet [27]. In this case, we group all amino acids not in the groups A or B into a neutral group, N. With this grouping, the correct classification rate for CB rises to approximately the rate achieved with the functional alphabet, although the Random test set performs better with other alphabet reductions. Charge would appear to be a significant force in the folding process.

Symbols 3 v4		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	95.94	449	0.00	0	1.71	8	2.35	11
	Test	92.54	186	0.00	0	2.49	5	4.98	10
	Total	**94.92**	**635**	**0.00**	**0**	**1.94**	**13**	**3.14**	**21**
CB	Train	5.41	16	72.64	215	12.84	38	9.12	27
	Test	5.51	7	69.29	88	18.11	23	7.09	9
	Total	**5.44**	**23**	**71.63**	**303**	**14.42**	**61**	**8.51**	**36**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	0.00	0	100.00	74	0.00	0
	Total	**0.00**	**0**	**0.00**	**0**	**100.00**	**245**	**0.00**	**0**
Random	Train	0.40	1	1.61	4	0.00	0	97.99	244
	Test	5.61	6	0.93	1	4.67	5	88.79	95
	Total	**1.97**	**7**	**1.40**	**5**	**1.40**	**5**	**95.22**	**339**

Table 4.8: Models using 3-letter "charge" alphabet

At the very lowest level of differentiation between types of amino acids, we grouped the amino acids into H (including Glycine) and P. The Random class initially would not train with this alphabet, but after an initial drop in probability training proceeded normally.

Symbols		Globin		CB		Kinase		Random	
2		%	#	%	#	%	#	%	#
Globin	Train	95.09	445	0.85	4	1.50	7	2.56	12
	Test	96.02	193	0.00	0	1.99	4	1.99	4
	Total	**95.37**	**638**	**0.60**	**4**	**1.64**	**11**	**2.39**	**16**
CB	Train	1.35	4	64.19	190	15.54	46	18.92	56
	Test	0.79	1	55.12	70	24.41	31	19.69	25
	Total	**1.18**	**5**	**61.47**	**260**	**18.20**	**77**	**19.15**	**81**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	2.70	2	97.30	72	0.00	0
	Total	**0.00**	**0**	**0.82**	**2**	**99.18**	**243**	**0.00**	**0**
Random	Train	0.00	0	2.01	5	0.00	0	97.99	244
	Test	0.00	0	9.35	10	4.67	5	85.98	92
	Total	**0.00**	**0**	**4.21**	**15**	**1.40**	**5**	**94.38**	**336**

Table 4.9: Models using 2-letter H/P alphabet

Results with non-biochemical 3-letter alphabets

In order to provide a control of sorts for alphabet reduction, we also considered alphabets reduced without regard to chemical or functional relationships. The results were somewhat unexpected. Surprisingly, the differentiation between Globin, Kinase and Random sequences was excellent for all three of these alphabets.

In the first such grouping, we formed groups with approximately the same codon redundancy, attempting to evenly split the hydrophobic and polar amino acids. In particular, we had {Ala (4), Phe (2), Ile (3), Lys (2), Asp (2), Ser (6), Trp (1)} for a total redundancy of 20; {Leu (6), Pro (4), Glu (2), Gln (2), Thr (4), Tyr (2)}; and {Val (4), Gly (4), Met (1), Arg (6), Cys (2), His (2), Asn (2)} with a total redundancy of 21.

Symbols		Globin		CB		Kinase		Random	
3 v1		%	#	%	#	%	#	%	#
Globin	Train	91.67	429	0.21	1	2.56	12	5.56	26
	Test	93.03	187	0.50	1	2.99	6	3.48	7
	Total	**92.08**	**616**	**0.30**	**2**	**2.69**	**18**	**4.93**	**33**
CB	Train	5.41	16	37.84	112	32.43	96	24.32	72
	Test	4.72	6	29.13	37	37.80	48	28.35	36
	Total	**5.20**	**22**	**35.22**	**149**	**34.04**	**144**	**25.53**	**108**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	0.00	0	98.65	73	1.35	1
	Total	**0.00**	**0**	**0.00**	**0**	**99.59**	**244**	**0.41**	**1**
Random	Train	0.00	0	1.61	4	0.00	0	98.39	245
	Test	0.00	0	4.67	5	3.74	4	91.59	98
	Total	**0.00**	**0**	**2.53**	**9**	**1.12**	**4**	**96.35**	**343**

Table 4.10: Codon redundancy alphabet

The models trained with this alphabet differentiate between Globin, Kinase and Random sequences very well, though CB sequences were assigned in approximately equal proportions to CB, Kinase and Random. Differentiation between CB and Globin is very good, although that in itself does not provide an adequate basis for a classification scheme. The CB model did train using this alphabet, which might explain the effective differentiation between CB and Globin, although the training does not appear to have been very effective in general for the CB model.

We next attempted to test the effect of Glycine by considering a three-letter alphabet in which Glycine is retained as a separate group and the remaining amino acids are randomly split into two groups of size 9 and 10. These groups are {Ala, Phe, His, Lys, Met, Ser, Thr, Val, Trp, Tyr} and {Cys, Asp, Glu, Ile, Leu, Asn, Pro, Gln, Arg}

This random grouping performed very similarly to the previous grouping. The CB

sequences are no longer differentiated from the Globins, however, perhaps because the CB model only trained for one iteration before decreasing steadily in average probability. These results could indicate that Glycine plays a large role in protein folding, but the next randomly generated alphabet seems to contradict that assertion.

Symbols		Globin		CB		Kinase		Random	
3 v2		%	#	%	#	%	#	%	#
Globin	Train	93.80	439	0.00	0	1.92	9	4.27	20
	Test	93.53	188	0.00	0	1.99	4	4.48	9
	Total	**93.72**	**627**	**0.00**	**0**	**1.94**	**13**	**4.33**	**29**
CB	Train	21.62	64	0.00	0	21.96	65	56.42	167
	Test	20.47	26	0.00	0	26.77	34	52.76	67
	Total	**21.28**	**90**	**0.00**	**0**	**23.40**	**99**	**55.32**	**234**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	0.00	0	100.00	74	0.00	0
	Total	**0.00**	**0**	**0.00**	**0**	**100.00**	**245**	**0.00**	**0**
Random	Train	0.40	1	0.00	0	0.00	0	99.60	248
	Test	1.87	2	0.00	0	8.41	9	89.72	96
	Total	**0.84**	**3**	**0.00**	**0**	**2.53**	**9**	**96.63**	**344**

Table 4.11: Random alphabet 1

The final non-biochemical grouping of amino acids was a random grouping into two groups of approximately equal size and a single group of size one (analogous to the Glycine group in the previous alphabet). The groups are {Cys, Glu, Gly, His, Ile, Lys, Met, Pro, Gln, Trp}, and {Ala, Asp, Phe, Leu, Asn, Arg, Thr, Val, Tyr} and {Ser}.

Except for a slight dip in the correct classification of the Random sequences, this alphabet gives results very similar to those with the previous alphabet. This might indicate that Glycine is not particularly important in the folding process. However, Serine's codon redundancy is six, so we may have inadvertently replaced one impor-

tant amino acid with another. It is more likely, though, that we have demonstrated that grouping amino acids has little impact on model performance. Those classes with high probability of seeing a certain amino acid at a certain position will simply train to emphasize the probability at that position of whichever group contains that amino acid.

Symbols		Globin		CB		Kinase		Random	
3 v3		%	#	%	#	%	#	%	#
Globin	Train	96.58	452	0.00	0	1.92	9	1.50	7
	Test	96.02	193	0.00	0	1.99	4	1.99	4
	Total	**96.41**	**645**	**0.00**	**0**	**1.94**	**13**	**1.64**	**11**
CB	Train	37.50	111	0.34	1	18.24	54	43.92	130
	Test	39.37	50	0.00	0	23.62	30	37.01	47
	Total	**38.06**	**161**	**0.24**	**1**	**19.86**	**84**	**41.84**	**177**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	0.00	0	100.00	74	0.00	0
	Total	**0.00**	**0**	**0.00**	**0**	**100.00**	**245**	**0.00**	**0**
Random	Train	2.01	5	0.00	0	1.61	4	96.39	240
	Test	7.48	8	0.00	0	10.28	11	82.24	88
	Total	**3.65**	**13**	**0.00**	**0**	**4.21**	**15**	**92.13**	**328**

Table 4.12: Random alphabet 2

4.2 Baldi update models

We trained models using the Baldi update procedure for the 20, 8, 5 and 2 letter alphabets and also for the 3-letter structural alphabet. Using a learning rate of 0.6, the average probability on the training sets was generally lower at convergence than for the Baum-Welsh (BW) models and each iteration had a smaller effect on average probability. This disparity was particularly noticeable for the Globin and CB classes, yet virtually non-existent for the Random class. For the full alphabet the

results were comparable to those with the BW update, but for alternate alphabets the results almost all resembled those with the full alphabet, showing poor recognition of the Kinase and Random test sets. Although recognition of CB remained good with each alphabet reduction, we would have liked to see improvement in the Kinase class.

Unlike the BW models, the number of *Match*'s in the best path for CB was as expected for each alphabet. Calculation of the best path, which is used for the Baldi update routine, does not involve division of the sequence into regions as with the BW update routine. The Baldi routine performs better overall on the CB class, indicating that we probably do not appropriately divide the sequences into segments for the BW update.

The results with the full alphabet are very similar to those with the other training method, although performance on Globin is somewhat reduced. When we consider the chemical alphabet, unlike the BW trained models, performance on Kinase remains poor on the test set and performance on the CB class remains good. The results with the Chemical alphabet resemble those for the full alphabet much more closely than we might expect given the difference in results between these two alphabets with the BW update.

Similarly, the 5-letter alphabet strongly resembles the full alphabet, with a small decrease across the board in sequence recognition. Like the chemical alphabet, we see neither the improved recognition of the Kinase test set nor the decreased recognition of CB sequences which we see with the BW updated models. In fact, most of the reduced alphabets behave very similarly with the Baldi updated models, not demonstrating the variation in performance shown with the BW updated models.

Symbols		Globin		CB		Kinase		Random	
20		%	#	%	#	%	#	%	#
Globin	Train	86.11	403	11.32	53	0.43	2	2.14	10
	Test	89.55	180	8.46	17	0.00	0	1.99	4
	Total	**87.14**	**583**	**10.46**	**70**	**0.30**	**2**	**2.09**	**14**
CB	Train	0.00	0	98.65	292	0.00	0	1.35	4
	Test	0.00	0	97.64	124	0.00	0	2.36	3
	Total	**0.00**	**0**	**98.35**	**416**	**0.00**	**0**	**1.65**	**7**
Kinase	Train	0.00	0	1.75	3	98.25	168	0.00	0
	Test	0.00	0	52.70	39	47.30	35	0.00	0
	Total	**0.00**	**0**	**17.14**	**42**	**82.86**	**203**	**0.00**	**0**
Random	Train	0.00	0	4.02	10	0.40	1	95.58	238
	Test	0.00	0	21.50	23	6.54	7	71.96	77
	Total	**0.00**	**0**	**9.27**	**33**	**2.25**	**8**	**88.48**	**315**

Table 4.13: Baldi update models, full alphabet

Symbols		Globin		CB		Kinase		Random	
8		%	#	%	#	%	#	%	#
Globin	Train	84.83	397	8.97	42	3.21	15	2.99	14
	Test	90.55	182	5.97	12	1.49	3	1.99	4
	Total	**86.55**	**579**	**8.07**	**54**	**2.69**	**18**	**2.69**	**18**
CB	Train	0.00	0	93.24	276	2.03	6	4.73	14
	Test	0.00	0	92.13	117	3.94	5	3.94	5
	Total	**0.00**	**0**	**92.91**	**393**	**2.60**	**11**	**4.49**	**19**
Kinase	Train	0.00	0	8.19	14	91.81	157	0.00	0
	Test	0.00	0	50.00	37	50.00	37	0.00	0
	Total	**0.00**	**0**	**20.82**	**51**	**79.18**	**194**	**0.00**	**0**
Random	Train	0.00	0	7.23	18	3.21	8	89.56	223
	Test	0.00	0	21.50	23	12.15	13	66.36	71
	Total	**0.00**	**0**	**11.52**	**41**	**5.90**	**21**	**82.58**	**294**

Table 4.14: Baldi update models, 8-letter chemical alphabet

Symbols		Globin		CB		Kinase		Random	
3 v5		%	#	%	#	%	#	%	#
Globin	Train	86.54	405	5.77	27	1.50	7	6.20	29
	Test	91.54	184	3.48	7	0.50	1	4.48	9
	Total	**88.04**	**589**	**5.08**	**34**	**1.20**	**8**	**5.68**	**38**
CB	Train	0.00	0	86.49	256	3.04	9	10.47	31
	Test	0.00	0	81.89	104	0.79	1	17.32	22
	Total	**0.00**	**0**	**85.11**	**360**	**2.36**	**10**	**12.53**	**53**
Kinase	Train	0.00	0	18.71	32	70.76	121	10.53	18
	Test	0.00	0	39.19	29	29.73	22	31.08	23
	Total	**0.00**	**0**	**24.90**	**61**	**58.37**	**143**	**16.73**	**41**
Random	Train	0.00	0	10.84	27	4.02	10	85.14	212
	Test	0.00	0	20.56	22	9.35	10	70.09	75
	Total	**0.00**	**0**	**13.76**	**49**	**5.62**	**20**	**80.62**	**287**

Table 4.15: Baldi update models, 3-letter structural alphabet

Symbols		Globin		CB		Kinase		Random	
5		%	#	%	#	%	#	%	#
Globin	Train	81.84	383	8.55	40	5.13	24	4.49	21
	Test	87.06	175	7.46	15	2.99	6	2.49	5
	Total	**83.41**	**558**	**8.22**	**55**	**4.48**	**30**	**3.89**	**26**
CB	Train	0.00	0	92.57	274	1.69	5	5.74	17
	Test	0.00	0	90.55	115	1.57	2	7.87	10
	Total	**0.00**	**0**	**91.96**	**389**	**1.65**	**7**	**6.38**	**27**
Kinase	Train	0.00	0	18.71	32	81.29	139	0.00	0
	Test	0.00	0	59.46	44	40.54	30	0.00	0
	Total	**0.00**	**0**	**31.02**	**76**	**68.98**	**169**	**0.00**	**0**
Random	Train	0.00	0	10.04	25	4.02	10	85.94	214
	Test	0.00	0	22.43	24	11.21	12	66.36	71
	Total	**0.00**	**0**	**13.76**	**49**	**6.18**	**22**	**80.06**	**285**

Table 4.16: Baldi update models, 5-letter alphabet

Symbols		Globin		CB		Kinase		Random	
2		%	#	%	#	%	#	%	#
Globin	Train	81.41	381	8.97	42	5.98	28	3.63	17
	Test	89.55	180	6.47	13	2.49	5	1.49	3
	Total	**83.86**	**561**	**8.22**	**55**	**4.93**	**33**	**2.99**	**20**
CB	Train	0.00	0	87.16	258	2.36	7	10.47	31
	Test	0.00	0	83.46	106	3.15	4	13.39	17
	Total	**0.00**	**0**	**86.05**	**364**	**2.60**	**11**	**11.35**	**48**
Kinase	Train	0.00	0	29.82	51	70.18	120	0.00	0
	Test	0.00	0	52.70	39	41.89	31	5.41	4
	Total	**0.00**	**0**	**36.73**	**90**	**61.63**	**151**	**1.63**	**4**
Random	Train	0.00	0	25.70	64	7.23	18	67.07	167
	Test	0.00	0	38.32	41	17.76	19	43.93	47
	Total	**0.00**	**0**	**29.49**	**105**	**10.39**	**37**	**60.11**	**214**

Table 4.17: Baldi update models, 2-letter alphabet

4.3 Amino acid composition

Training of the models for amino acid composition profiles was very fast, due to a combination of factors. The short length of the model and the processed sequences (19) meant very little time per training iteration. In addition, convergence occurred for all models in at most fifteen iterations. Thus, we could easily examine the effect of variation of the number of mixtures on model performance.

Since sequence length does not matter in this representation of sequences, we used the same training and test data sets for the Globin, CB and Random classes as for the symbolic representations. In addition, we used all available Kinase sequences in the length range [100,1000] (430) with a training set of size 300.

We first considered the single mixture model for amino acid composition profiles. The model cannot really train in the single-mixture continuous distribution model

because without multiple mixtures, there are no variable parameters to adjust. However, consideration of the single mixture case gives us the evaluation of the model essentially using the sample statistics of the training set.

AA Comp 1 Mixture		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	90.17	422	4.70	22	2.78	13	2.35	11
	Test	92.04	185	4.98	10	2.49	5	0.50	1
	Total	**90.73**	**607**	**4.78**	**32**	**2.69**	**18**	**1.79**	**12**
CB	Train	1.01	3	70.27	208	15.54	46	13.18	39
	Test	0.00	0	74.02	94	12.60	16	13.39	17
	Total	**0.71**	**3**	**71.39**	**302**	**14.66**	**62**	**13.24**	**56**
Kinase	Train	2.77	8	9.69	28	71.63	207	15.92	46
	Test	3.20	4	9.60	12	58.40	73	28.80	36
	Total	**2.90**	**12**	**9.66**	**40**	**67.63**	**280**	**19.81**	**82**
Random	Train	0.00	0	13.65	34	9.24	23	77.11	192
	Test	0.00	0	12.15	13	7.48	8	80.37	86
	Total	**0.00**	**0**	**13.20**	**47**	**8.71**	**31**	**78.09**	**278**

Table 4.18: Amino acid composition models, single mixture

The single mixture case for amino acid composition performs moderately well. Clearly some pattern detection is occurring, but the only class really successfully distinguished is Globin. The Globin sequences perform better on the Globin model than on any other and very few sequences from other classes do, giving a low false positive rate.

AA Comp 5 Mixtures		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	89.96	421	3.63	17	4.27	20	2.14	10
	Test	92.54	186	1.99	4	2.99	6	2.49	5
	Total	**90.73**	**607**	**3.14**	**21**	**3.89**	**26**	**2.24**	**15**
CB	Train	0.34	1	71.96	213	18.24	54	9.46	28
	Test	0.00	0	74.02	94	14.96	19	11.02	14
	Total	**0.24**	**1**	**72.58**	**307**	**17.26**	**73**	**9.93**	**42**
Kinase	Train	0.35	1	7.61	22	85.47	247	6.57	19
	Test	0.80	1	5.60	7	77.60	97	16.00	20
	Total	**0.48**	**2**	**7.00**	**29**	**83.09**	**344**	**9.42**	**39**
Random	Train	0.80	2	9.24	23	14.46	36	75.50	188
	Test	0.00	0	9.35	10	10.28	11	80.37	86
	Total	**0.56**	**2**	**9.27**	**33**	**13.20**	**47**	**76.97**	**274**

Table 4.19: Amino acid composition models, 5 mixtures

Increasing the number of mixtures to five improves results somewhat. While this would not be considered a very reliable classifier, these results indicate that amino acid composition definitely correlates to structural class. The average rate of correct classification over all samples tested was 82.28%. These results are not quite as strong with these definitions of structural classes as Chou and Zhang achieved with their definitions.

AA Comp 10 Mixtures		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	89.32	418	4.27	20	4.27	20	2.14	10
	Test	92.54	186	1.99	4	2.99	6	2.49	5
	Total	**90.28**	**604**	**3.59**	**24**	**3.89**	**26**	**2.24**	**15**
CB	Train	0.34	1	70.95	210	18.24	54	10.47	31
	Test	0.00	0	74.02	94	14.96	19	11.02	14
	Total	**0.24**	**1**	**71.87**	**304**	**17.26**	**73**	**10.64**	**45**
Kinase	Train	0.35	1	7.61	22	85.12	246	6.92	20
	Test	0.80	1	5.60	7	77.60	97	16.00	20
	Total	**0.48**	**2**	**7.00**	**29**	**82.85**	**343**	**9.66**	**40**
Random	Train	0.80	2	8.43	21	14.46	36	76.31	190
	Test	0.00	0	9.35	10	10.28	11	80.37	86
	Total	**0.56**	**2**	**8.71**	**31**	**13.20**	**47**	**77.53**	**276**

Table 4.20: Amino acid composition models, 10 mixtures

Increasing the number of mixtures to ten marginally decreases overall performance, but essentially leaves the results the same. Note that the training and test sets perform with comparable levels of accuracy. This probably indicates the existence of generalizable characteristics of each class within this representation of the sequences.

We also tested this type of model with 20, 25, 30, 35, 40 and 50 mixtures; the model performed with average accurate classification between 81.95% and 82.12%. After the slight dip in performance with 10 mixtures, the model appeared to stabilize at a level just slightly below the level of performance with 5 mixtures. Thus, for this type of sequence representation, 5 mixtures appears adequate.

4.4 Transformed property profiles

To prepare the property profiles, we first read the sequence data, replacing the amino acid code with its value on a particular property and padding the sequence length to a power of two with zeros. After performing a fast Fourier transform, we consider the average power over the training set for each Fourier component and retain the n_sig most powerful components to represent the sequences.

4.4.1 Data sets

For this representation of sequences, we restricted Globin, CB and Random sequences to the length range [128,256] (the power-of-two range containing the largest number of samples for those classes), using 651, 167, and 188 sequences respectively. For the Kinase group, we chose sequences in the range [256,512] for a total of 245 sequences. The CB training set for these experiments may not have been of adequate size to generalize to the test sets. As a further test of the impact of training set size on generalization, we tested Kinase sequences in the range [128,256] with the very small data set (79 total sequences) available.

Since sequences for different classes may come from different length ranges, we replace non-existent components with zeros. For sequences from the length range $[128, 256]$, the number of components with distinct powers is 128, but for sequences in the range $[256, 512]$, there are 256 distinct values. If, for those longer classes, a component with index greater than 128 is selected to represent the sequences, shorter sequences (from other classes) will have zeros in that position when being represented to the model. This is simply equivalent to saying that the shorter sequences do not have higher frequency components in their transforms.

4.4.2 Single versus multiple properties

We constructed models for each of the properties of Hydrophobicity (PONNU), Accessible Surface Area, van der Waals volume, Charge, Hydrophobicity (ROSEF) and

Hydrophobicity (PRIFT). While these models perform reasonably well individually, considering sets of models for each class, trained with different properties, might compensate for imperfections in any individual model.

Initially, we considered all six properties at once, assigning a sequence to a class if it scored higher on a the greatest number of models for that class. For example, if a sequence scores highest ("wins") on the Globin models for Hydrophobicity, van der Waals volume, and Hydrophobicity (ROSEF), wins on the Calcium-Binding model for Charge, wins on the Kinase model for Accessible Surface area, and wins on the Random model for Hydrophobicity (PRIFT), we would assign that sequence to the Globin set. The size of individual differences in scores is not considered, although it is interesting to note that on average the difference is much smaller for sequences which are mis-classified. We have found that performance variation between properties is slight and considering the properties simultaneously in this way provides an "average" case.

4.4.3 Models for each property considered simultaneously

128 significant components

We again start by considering the single mixture case. It is surprisingly effective at distinguishing between Globin and Kinase sequences. For the Calcium-Binding class results on the training set are good, but have poor generalization to the test set.

128 Comp, 1 Mix		Globin		CB		Kinase		Random		Tied	
prop 1 2 3 4 5 6		%	#	%	#	%	#	%	#	%	#
Globin	Train	88.79	404	4.84	22	0.00	0	3.52	16	2.86	13
	Test	93.37	183	3.06	6	0.00	0	1.53	3	2.04	4
	Total	**90.17**	**587**	**4.30**	**28**	**0.00**	**0**	**2.92**	**19**	**2.61**	**17**
CB	Train	0.00	0	87.93	102	0.00	0	6.90	8	5.17	6
	Test	0.00	0	60.78	31	0.00	0	35.29	18	3.92	2
	Total	**0.00**	**0**	**79.64**	**133**	**0.00**	**0**	**15.57**	**26**	**4.79**	**8**
Kinase	Train	0.00	0	0.00	0	94.15	161	2.92	5	2.92	5
	Test	0.00	0	0.00	0	86.49	64	4.05	3	9.46	7
	Total	**0.00**	**0**	**0.00**	**0**	**91.84**	**225**	**3.27**	**8**	**4.90**	**12**
Random	Train	1.25	2	3.75	6	0.00	0	91.88	147	3.12	5
	Test	1.45	1	8.70	6	0.00	0	86.96	60	2.90	2
	Total	**1.31**	**3**	**5.24**	**12**	**0.00**	**0**	**90.39**	**207**	**3.06**	**7**

Table 4.21: 6-property comparison, single mixture, 128 components

We tested results with 5, 10, 20 and 30 mixtures and average performance increased steadily with the number of mixtures. We will present results with 30 mixtures for the remainder of the discussion because that is the largest number of mixtures we tested and this construction provided the best results. Performance might improve still more with an increased number of mixtures.

128 Comp, 30 Mix		Globin		CB		Kinase		Random		Tied	
prop 1 2 3 4 5 6		%	#	%	#	%	#	%	#	%	#
Globin	Train	91.21	415	0.00	0	0.00	0	5.49	25	3.30	15
	Test	90.31	177	0.00	0	0.00	0	5.10	10	4.59	9
	Total	**90.94**	**592**	**0.00**	**0**	**0.00**	**0**	**5.38**	**35**	**3.69**	**24**
CB	Train	0.00	0	99.14	115	0.00	0	0.86	1	0.00	0
	Test	0.00	0	49.02	25	0.00	0	45.10	23	5.88	3
	Total	**0.00**	**0**	**83.83**	**140**	**0.00**	**0**	**14.37**	**24**	**1.80**	**3**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0	0.00	0
	Test	0.00	0	0.00	0	98.65	73	0.00	0	1.35	1
	Total	**0.00**	**0**	**0.00**	**0**	**99.59**	**244**	**0.00**	**0**	**0.41**	**1**
Random	Train	0.00	0	0.63	1	0.00	0	99.38	159	0.00	0
	Test	0.00	0	0.00	0	0.00	0	98.55	68	1.45	1
	Total	**0.00**	**0**	**0.44**	**1**	**0.00**	**0**	**99.13**	**227**	**0.44**	**1**

Table 4.22: 6-property comparison, 30 mixtures, 128 significant components

With 30 mixtures, we achieved an average correct classification of 93.11% when considering all properties at once. The fact that the CB class did not perform nearly that well on its test set may be a testament to the fact that the data set was simply too small.

Short Kinase sequences

For completeness, we tested Kinase sequences in the range [128,256]. The class has a small data set in this length range and we can clearly see the impact of this small set size on performance. While this class achieves 100% recognition of its training set, it virtually fails to recognize the test set. This illustrates directly the impact of training set size on performance.

128 Comp, 30 Mix		Globin		CB		Kinase		Random		Tied	
prop 1 2 3 4 5 6		%	#	%	#	%	#	%	#	%	#
Globin	Train	91.21	415	0.00	0	0.00	0	5.49	25	3.30	15
	Test	90.31	177	0.00	0	0.00	0	5.10	10	4.59	9
	Total	**90.94**	**592**	**0.00**	**0**	**0.00**	**0**	**5.38**	**35**	**3.69**	**24**
CB	Train	0.00	0	99.14	115	0.00	0	0.86	1	0.00	0
	Test	0.00	0	49.02	25	0.00	0	45.10	23	5.88	3
	Total	**0.00**	**0**	**83.83**	**140**	**0.00**	**0**	**14.37**	**24**	**1.80**	**3**
Kinase	Train	0.00	0	0.00	0	100.00	55	0.00	0	0.00	0
	Test	0.00	0	0.00	0	0.00	0	95.83	23	4.17	1
	Total	**0.00**	**0**	**0.00**	**0**	**69.62**	**55**	**29.11**	**23**	**1.27**	**1**
Random	Train	0.00	0	0.63	1	0.00	0	99.38	159	0.00	0
	Test	0.00	0	0.00	0	0.00	0	98.55	68	1.45	1
	Total	**0.00**	**0**	**0.44**	**1**	**0.00**	**0**	**99.13**	**227**	**0.44**	**1**

Table 4.23: 6-property model, 30 mixtures, short Kinase sequences

64 significant components

We initially retained 128 significant components from the transformed property profiles because that was the maximum number available for consideration of sequences in the range [128,256], but can we reduce that number and still differentiate between classes? In fact, retaining 64 components decreases performance dramatically. The only classes really distinguished from the others are the Random and Kinase classes. This might be useful to distinguish between Kinase and "other" sequences, but for no finer differentiation.

64 Comp, 30 Mix		Globin		CB		Kinase		Random		Tied	
prop 1 2 3 4 5 6		%	#	%	#	%	#	%	#	%	#
Globin	Train	49.67	226	0.00	0	0.00	0	41.10	187	9.23	42
	Test	48.47	95	0.51	1	0.00	0	39.80	78	11.22	22
	Total	**49.31**	**321**	**0.15**	**1**	**0.00**	**0**	**40.71**	**265**	**9.83**	**64**
CB	Train	0.00	0	68.97	80	0.00	0	12.93	15	18.10	21
	Test	0.00	0	29.41	15	0.00	0	58.82	30	11.76	6
	Total	**0.00**	**0**	**56.89**	**95**	**0.00**	**0**	**26.95**	**45**	**16.17**	**27**
Kinase	Train	0.00	0	0.00	0	86.55	148	3.51	6	9.94	17
	Test	0.00	0	0.00	0	78.38	58	4.05	3	17.57	13
	Total	**0.00**	**0**	**0.00**	**0**	**84.08**	**206**	**3.67**	**9**	**12.24**	**30**
Random	Train	0.00	0	0.00	0	0.00	0	97.50	156	2.50	4
	Test	0.00	0	1.45	1	0.00	0	86.96	60	11.59	8
	Total	**0.00**	**0**	**0.44**	**1**	**0.00**	**0**	**94.32**	**216**	**5.24**	**12**

Table 4.24: 6-property model, 30 mixtures, 64 significant components

32 significant components

Reduction to 32 components gives results little better than random assignment to classes. The overall average rate of correct classification is only 28.95%, but most results are far worse. The peak recognition is of the Random set. The results on the Random set can be explained by the fact that 68.49% of the non-Random sequences were classified as Random–a very high false positive rate. So the Random set scored highest on the Random model and so did most of the other sequences.

32 Comp, 30 Mix		Globin		CB		Kinase		Random		Tied	
prop 1 2 3 4 5 6		%	#	%	#	%	#	%	#	%	#
Globin	Train	6.15	28	0.00	0	0.00	0	87.69	399	6.15	28
	Test	6.63	13	0.51	1	0.00	0	87.24	171	5.61	11
	Total	**6.30**	**41**	**0.15**	**1**	**0.00**	**0**	**87.56**	**570**	**5.99**	**39**
CB	Train	0.00	0	22.41	26	0.00	0	56.03	65	21.55	25
	Test	0.00	0	9.80	5	0.00	0	80.39	41	9.80	5
	Total	**0.00**	**0**	**18.56**	**31**	**0.00**	**0**	**63.47**	**106**	**17.96**	**30**
Kinase	Train	0.58	1	1.17	2	57.89	99	20.47	35	19.88	34
	Test	2.70	2	4.05	3	47.30	35	22.97	17	22.97	17
	Total	**1.22**	**3**	**2.04**	**5**	**54.69**	**134**	**21.22**	**52**	**20.82**	**51**
Random	Train	5.63	9	8.75	14	0.00	0	78.12	125	7.50	12
	Test	8.70	6	13.04	9	0.00	0	62.32	43	15.94	11
	Total	**6.55**	**15**	**10.04**	**23**	**0.00**	**0**	**73.36**	**168**	**10.04**	**23**

Table 4.25: 6-property model, 30 mixtures, 32 significant components

4.4.4 Individual properties

On the whole, models for each of the properties considered performed at comparable levels. The property showing the highest average correct classification for the 30-mixture, 128 significant component case was hydrophobicity (PRIFT), with 1222 of the 1292 sequences considered correctly classified (94.58%).

128 Comp, 30 Mix prop 1		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	94.73	431	0.66	3	0.00	0	4.62	21
	Test	92.86	182	0.00	0	0.00	0	7.14	14
	Total	**94.16**	**613**	**0.46**	**3**	**0.00**	**0**	**5.38**	**35**
CB	Train	0.00	0	99.14	115	0.00	0	0.86	1
	Test	3.92	2	50.98	26	5.88	3	39.22	20
	Total	**1.20**	**2**	**84.43**	**141**	**1.80**	**3**	**12.57**	**21**
Kinase	Train	0.00	0	0.00	0	98.25	168	1.75	3
	Test	0.00	0	0.00	0	93.24	69	6.76	5
	Total	**0.00**	**0**	**0.00**	**0**	**96.73**	**237**	**3.27**	**8**
Random	Train	0.00	0	0.63	1	0.00	0	99.38	159
	Test	2.90	2	2.90	2	4.35	3	89.86	62
	Total	**0.87**	**2**	**1.31**	**3**	**1.31**	**3**	**96.51**	**221**

Table 4.26: Hydrophobicity (PONNU)

128 Comp, 30 Mix prop 2		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	88.35	402	0.22	1	0.00	0	11.43	52
	Test	89.29	175	1.02	2	0.51	1	9.18	18
	Total	**88.63**	**577**	**0.46**	**3**	**0.15**	**1**	**10.75**	**70**
CB	Train	0.00	0	97.41	113	0.00	0	2.59	3
	Test	0.00	0	54.90	28	1.96	1	43.14	22
	Total	**0.00**	**0**	**84.43**	**141**	**0.60**	**1**	**14.97**	**25**
Kinase	Train	0.00	0	0.00	0	98.83	169	1.17	2
	Test	0.00	0	0.00	0	98.65	73	1.35	1
	Total	**0.00**	**0**	**0.00**	**0**	**98.78**	**242**	**1.22**	**3**
Random	Train	0.00	0	0.00	0	0.00	0	100.00	160
	Test	0.00	0	2.90	2	4.35	3	92.75	64
	Total	**0.00**	**0**	**0.87**	**2**	**1.31**	**3**	**97.82**	**224**

Table 4.27: Accessible surface area

128 Comp, 30 Mix prop 3		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	89.01	405	0.00	0	0.00	0	10.99	50
	Test	90.82	178	0.51	1	0.00	0	8.67	17
	Total	**89.55**	**583**	**0.15**	**1**	**0.00**	**0**	**10.29**	**67**
CB	Train	0.00	0	99.14	115	0.00	0	0.86	1
	Test	3.92	2	49.02	25	1.96	1	45.10	23
	Total	**1.20**	**2**	**83.83**	**140**	**0.60**	**1**	**14.37**	**24**
Kinase	Train	0.00	0	0.00	0	98.25	168	1.75	3
	Test	0.00	0	0.00	0	98.65	73	1.35	1
	Total	**0.00**	**0**	**0.00**	**0**	**98.37**	**241**	**1.63**	**4**
Random	Train	0.00	0	0.63	1	0.00	0	99.38	159
	Test	5.80	4	0.00	0	1.45	1	92.75	64
	Total	**1.75**	**4**	**0.44**	**1**	**0.44**	**1**	**97.38**	**223**

Table 4.28: van der Waals Volume

128 Comp, 30 Mix prop 4		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	87.91	400	0.00	0	0.22	1	11.87	54
	Test	85.71	168	0.00	0	0.51	1	13.78	27
	Total	**87.25**	**568**	**0.00**	**0**	**0.31**	**2**	**12.44**	**81**
CB	Train	0.00	0	96.55	112	0.00	0	3.45	4
	Test	0.00	0	45.10	23	7.84	4	47.06	24
	Total	**0.00**	**0**	**80.84**	**135**	**2.40**	**4**	**16.77**	**28**
Kinase	Train	0.00	0	0.00	0	97.66	167	2.34	4
	Test	0.00	0	0.00	0	98.65	73	1.35	1
	Total	**0.00**	**0**	**0.00**	**0**	**97.96**	**240**	**2.04**	**5**
Random	Train	0.63	1	0.63	1	0.00	0	98.75	158
	Test	10.14	7	0.00	0	4.35	3	85.51	59
	Total	**3.49**	**8**	**0.44**	**1**	**1.31**	**3**	**94.76**	**217**

Table 4.29: Charge

128 Comp, 30 Mix prop 5		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	92.09	419	0.88	4	0.00	0	7.03	32
	Test	93.37	183	1.53	3	0.00	0	5.10	10
	Total	**92.47**	**602**	**1.08**	**7**	**0.00**	**0**	**6.45**	**42**
CB	Train	0.00	0	99.14	115	0.00	0	0.86	1
	Test	1.96	1	47.06	24	0.00	0	50.98	26
	Total	**0.60**	**1**	**83.23**	**139**	**0.00**	**0**	**16.17**	**27**
Kinase	Train	0.00	0	0.00	0	95.91	164	4.09	7
	Test	0.00	0	0.00	0	91.89	68	8.11	6
	Total	**0.00**	**0**	**0.00**	**0**	**94.69**	**232**	**5.31**	**13**
Random	Train	0.00	0	0.63	1	0.00	0	99.38	159
	Test	0.00	0	1.45	1	1.45	1	97.10	67
	Total	**0.00**	**0**	**0.87**	**2**	**0.44**	**1**	**98.69**	**226**

Table 4.30: Hydrophobicity (ROSEF)

128 Comp, 30 Mix prop 6		Globin		CB		Kinase		Random	
		%	#	%	#	%	#	%	#
Globin	Train	92.97	423	1.54	7	0.00	0	5.49	25
	Test	93.37	183	1.02	2	0.00	0	5.61	11
	Total	**93.09**	**606**	**1.38**	**9**	**0.00**	**0**	**5.53**	**36**
CB	Train	0.00	0	99.14	115	0.00	0	0.86	1
	Test	0.00	0	66.67	34	0.00	0	33.33	17
	Total	**0.00**	**0**	**89.22**	**149**	**0.00**	**0**	**10.78**	**18**
Kinase	Train	0.00	0	0.00	0	100.00	171	0.00	0
	Test	0.00	0	0.00	0	93.24	69	6.76	5
	Total	**0.00**	**0**	**0.00**	**0**	**97.96**	**240**	**2.04**	**5**
Random	Train	0.00	0	0.63	1	0.00	0	99.38	159
	Test	0.00	0	1.45	1	0.00	0	98.55	68
	Total	**0.00**	**0**	**0.87**	**2**	**0.00**	**0**	**99.13**	**227**

Table 4.31: Hydrophobicity (PRIFT)

Chapter 5 Discussion

5.1 Symbolic models

The symbolic model using the full twenty-letter alphabet differentiated very well between the structural classes we modeled. However, we can effectively capture structural class information using models with fewer symbols. This indicates redundancy in the protein alphabet, as expected.

In considering symbolic alphabet reductions, we tried to find a particular grouping of the amino acids demonstrating exceptionally good ability to differentiate between structural classes with our model format. Alphabet reductions potentially increase sequence similarity, and a high level of similarity would lead to increased recognition of sequences. In searching for the best alphabet reduction to a specific number of symbols, we used groupings based on properties known to be significant in the folding process.

While members of a class might have a variety of amino acids appearing from a particular node and state of the model in their best paths, perhaps those amino acids are all (for example) hydrophobic. Using an alphabet which groups together hydrophobic amino acids would lead to increased sequence similarity at that node and state. Though the primary goal of alphabet reduction is to demonstrate that we do not require twenty symbols to capture the structural class information contained in an amino acid sequence, another objective is identification (or verification) of significant forces in the folding process. If the highest probability amino acids at a particular node and state of the model all share a common biochemical property, we can deduce that the important force at that position relates to that property.

Our experiments with reduced alphabets indicate that some groupings of amino acids lead to better differentiation between structural classes than others. The greatest performance variations occur for the CB class and, thus, the results are somewhat

confused by the difficulty with multi-domain sequences. However, considering classifications between all four classes, between Globin, Kinase and Random, and between Globin, CB and Random, some alphabets ranked fairly high in average correct classification for all three cases.

Overall, the full alphabet performed the best. However, three groupings appeared to lead to consistently better results than the others: the 3-letter "structural" alphabet, the 4-letter "functional" alphabet, and the 8-letter "chemical" alphabet. These groupings are followed by the remainder of the groupings based on hydrophobicity and polarity, which are followed by the non-biochemical/random groupings.

The performance of the structural alphabet is surprisingly good, considering that we have reduced the number of symbols from twenty to three. The structural alphabet groups amino acids based on preference for general position within the native fold. Using this alphabet appears to lead to some amount of increased sequence identity within a structural class. This grouping does not give us much indication about the forces driving the folding process since it is derived from observation, rather than from biochemical analysis. But, whatever the derivation, we appear to capture the majority of structural class information. This indicates significant redundancy in the protein alphabet.

Hydrophobic amino acids tend to prefer the interior of a folded protein and charged amino acids prefer the exterior, so in a way the functional alphabet is closely related to the structural one. The performance of the functional alphabet ($H'/A/B/U$) was somewhat better than the nearly identical 4-letter $H/P_0/P_-/P_+$ alphabet, indicating that the three amino acid re-assignments probably gave a more appropriate grouping. However, the difference in the classification accuracies was small enough that more tests would be required to fully assess the relative merits of these two groupings.

The performance of the chemical alphabet may have to do with the appropriate grouping of amino acids, or it may have to do with the fact that it was the largest reduced alphabet we tested and thus contained the largest number of bits of information. However, two of the groups considered were A and B, as for the functional alphabet, and the performance of the functional alphabet indicated that these group-

ings had merit in structure prediction. Perhaps we see good performance due to the appropriateness of that grouping.

The number of symbols used to represent a protein sequence does appear to matter somewhat in differentiating between structural classes. But we have found that a really dramatic alphabet reduction, based on known folding principles or known amino acid tendencies, can work almost as well as the full alphabet. The variations between the four letter alphabets definitely indicate that some are closer to "optimal" (in the sense of containing all available structural class information) than others. Appropriateness of grouping appears to matter nearly as much as alphabet size, since the 3-letter structural alphabet out-performs the 5-letter $H/P_0/P_-/P_+/G$ alphabet. We cannot, based on our results, assert that one set of biochemical properties has emerged as the most significant. However, we have shown that sequence recognition is possible with fewer than twenty symbols, given an appropriate set of biochemical properties to group amino acids.

While a biochemically rational grouping of amino acids can surely lead to heightened model performance, other groupings do not perform quite as poorly as we would expect. However, the non-biochemically reduced alphabets (codon redundancy and random groupings) do perform consistently worse than the others, even if not by a large margin. The non-biochemical alphabets very effectively differentiate between Globin, Kinase and Random sequences, achieving average correct classification rates from 95.43% to 95.91% when considering only these three classes, but each fails completely on the Calcium-Binding class.

The differentiation between the Globin, Kinase and Random classes strongly indicates the existence of a small set of high probability amino acids at each node for these classes. When we reduce the alphabet, the group containing the majority of these high probability amino acids may simply become the high probability group/symbol. Perhaps the Globin and Kinase classes start with a fair degree of sequence identity, thus retaining some amount of sequence identity in any alphabet reduction.

5.1.1 Multi-domain sequences

All classification errors of Kinase and Random sequences with the twenty-symbol alphabet erroneously assign the sequences to CB. When we reduce the alphabet, the CB models weaken and appear to assign relatively low values to the majority of sequences. This majority unfortunately includes some CB sequences.

The difficulty we have had with CB sequences most likely has to do with our treatment of multi-domain sequences with the BW update routine. Globin is theoretically a multi-domain class as well, but we used very few long Globin sequences. A much larger fraction of the CB sequences used are multi-domain, so any difficulties with using multi-domain sequences in this model will primarily affect the CB classifications.

Without comparison to the Kinase class, CB performs quite well on several of the reduced alphabets, with correct rates of classification of CB sequences never falling much below 75% for any of the biochemical reductions. The inclusion of the Kinase class may highlight the weakness of our use of multi-domain sequences by offering longer CB sequences the option of a longer model: the CB model consists of 100 nodes, while the Kinase model is 478 nodes long.

Reduction of the alphabet size corresponds to a larger number of *Match* states in the best path for most CB sequences. This indicates a greater degree of alignment of the sequence to the concatenated models. The fact that this "optimal" alignment to the CB model has lower probability than that to another (incorrect) model indicates a serious problem. Our treatment of multi-domain sequences entails hypothesis of probable alignment between the concatenated models and the sequence. In dividing the sequence into segments we assert that the d^{th} segment was probably generated by the d^{th} or $(d+1)^{st}$ model. We may have erred in the definition of the d^{th} segment, or perhaps in trying to treat the sequence in segments at all. The Baldi update routine uses the best path and does not divide a multi-domain sequence into segments.

5.1.2 Baum-Welch vs. Baldi

In most cases and for most classes, the BW update routine found a set of model parameters giving both higher average log of probability and higher average probability than the Baldi update routine. Convergence with both update methods occurred within a comparable number of iterations. For the Globin and Kinase classes, the BW update routine gave significantly higher probability and log probability on each alphabet tested. For each alphabet both of the above criteria were comparable on the Random class, but the CB results were mixed.

While the average probability of the CB training set was many orders of magnitude higher for each alphabet with the BW update routine, the average log of probability was significantly higher with the Baldi update routine. With such small probabilities, the average log of probability is a much more robust measure than the average probability. While the BW update led to some number of relatively high probability sequences which dominated the value of the average probability, the Baldi update appears to have achieved a higher level of probability for a greater number of sequences. This, again, most likely relates to the large number of multi-domain CB sequences.

Although the Baldi update outperforms the BW update for multi-domain sequences, it performs relatively poorly on the Globin and Kinase classes. Our results suggest that this routine is more prone to local extrema than the BW update method. Perhaps this is because the Baldi routine relies exclusively on the current optimal path through the model, while the BW routine relies on values calculated at a larger number of nodes and states.

5.2 Continuous observation models

For the symbolic alphabet we considered alphabet reductions and examined the effectiveness of several amino acid properties at differentiating, with this model construction, between structural classes. To examine the effect of amino acid properties with a real vector representation, we use the transformed property profiles. And to

examine a property of the entire sequence, we consider the amino acid composition.

We have found that representing a sequence with a transformed property profile leads to excellent differentiation between structural classes. When we construct a property profile, we translate the twenty letter amino acid alphabet into a twenty letter alphabet of floats (or three letters for charge). In representing the sequence with these values, we limit the information in the sequence to that about the property being modeled, but we clearly retain adequate information to effectively differentiate between structural classes. Although we did not find a single "most" valuable property, we can assert that each of the amino acid properties considered is strongly relevant to fold type. Amino acid composition's relationship to fold is more debatable, as the results with that sequence property were not very good.

5.2.1 Mixture variations

When we initialize the continuous observation models, we set the mixture parameters to the training set statistics, with variance variations if we have more than one mixture. For the single mixture models, no training actually occurs because the mixture parameters are the only variable model parameters when we eliminate alignment. Thus, evaluating performance of a sequence on a single mixture model is essentially a use of class statistics, since the training set statistics are presumably very similar. The novelty of this approach lies in the sequence representation: by using a processed representation, we eliminate the need to consider sequence alignment.

The single mixture model, while conceptually similar to other uses of class statistics for protein classification, has the advantage of not requiring alignment considerations. With multiple mixture models, we have an additional advantage with an increased level of flexibility. Not only does no quantity need to match another exactly (as with symbolic representations), there are even a variety of distributions (mixtures) which may have generated the quantity. The weighting given by the mixture coefficients allows emphasis on more than one distribution at a position. This addresses the potential for variations within a class in a way that a single set of statistics does

not.

5.2.2 Amino acid composition

The single mixture case appears to detect patterns within structural classes. However, incorporation of additional mixtures improves performance on the Kinase class considerably. The fact that increasing the number of mixtures beyond five has little effect indicates that we may have reached the limits of effectiveness for this sequence representation. The short training time (less than 15 iterations in all cases) may also reflect the relatively small amount of information contained in the amino acid composition vector. The results with this representation are interesting, though not as stellar for these class definitions as for Chou and Zhang's. These models clearly detect some level of pattern within our structural classes and generalize as well as possible to the test set.

5.2.3 Transformed property profiles

While the single mixture case gives results that are quite good (with an average correct classification rate of 89.16% for the 128 component/six property comparison), increasing the number of mixtures steadily improves performance (to 93.11% for the above with 30 mixtures). It appears that the transformed property profiles contain more complex sequence information than the amino acid composition vectors. This representation requires a larger number of mixtures to model well and requires more iterations for training. This more complex representation demands more of the model architecture and the training process than the amino acid composition does, but performance improves with this complexity.

Just as we hoped to find a particularly good grouping of amino acids for the symbolic model, we would have liked to see a particular amino acid property emerge as the best predictor of structural class. This did not happen, but only because all of the amino acid properties we considered gave very good results. It is not surprising that using each of these properties led to a model which could differentiate between

classes. We considered these properties because they are known to relate to folding. Hydrophobic amino acids strongly prefer the interior of a folded protein and charged amino acids strongly prefer the exterior.

The comparable results with the different properties may be a consequence of the relationships between them. Accessible surface area and van der Waals volume correlate strongly to hydrophobicity and we also used multiple hydrophobicity scales. Charge is essentially uncorrelated to hydrophobicity and the performance of the transformed charge profiles is surprisingly good. The charge alphabet contains only three symbols, so we might not expect such a high level of differentiation. However, the distribution of charged amino acids along the protein sequence relates to their three-dimensional distribution, which, in turn, relates strongly to the function of the protein. The results with the symbolic structural alphabet indicate that three symbols can adequately capture the class information in a sequence and the charge results reinforce that conclusion.

The only data set we considered which was not extremely well recognized by this type of model was the CB test set. This probably has to do with the size of the CB training set, which contains somewhat fewer sequences than those for the other classes. As we can see so clearly with the small set of short Kinase sequences tested, sample set size has a large effect on generalization to the test set. The CB data set for this sequence representation appears to be of marginal size: we see some generalization to the test set, but not as much as for the classes with larger training sets.

We would have liked to see more effective cooperation between models for different properties. In theory, a set of models might compensate for defects in each individual model, but in actuality several sequences are mis-classified by a majority of the models. These sequences should probably be examined more carefully for class membership. As another, possibly more effective, approach to the use of multiple properties, we could construct multi-dimensional property profiles–matrices of real numbers–and perform multi-dimensional transforms.

5.3 Conclusion

We have found that left-to-right Hidden Markov Models are very well suited to the task of differentiating between structural/functional classes of proteins, both with symbolic and with real vector representations. While this suitability had already been demonstrated for full alphabet, symbolic representations, we have shown that reduced alphabets can work nearly as well. This demonstrates the great redundancy in the protein alphabet.

The modifications we made to the model to use real vector representations give us a direct analogy to the problem of speech recognition. While amino acid composition vectors used with this model give mediocre results, transformed property profiles show excellent differentiation between structural classes. These representations effectively capture the structural class information contained within the protein sequence, while completely eliminating the need for alignment considerations. This appears to be a very promising basis for a structural/functional classification scheme.

Bibliography

[1] Pierre Baldi and Yves Chauvin. Smooth on-line learning algorithms for hidden Markov models. *Neural Computation*, 6:307–318, 1994.

[2] Pierre Baldi, Yves Chauvin, Tim Hunkapiller, and Marcella A. McClure. Hidden Markov models of biological primary sequence information. *Proc. Natl. Acad. Sci. USA*, 91:1059–1063, February 1994.

[3] J. Bascle, T. Garel, and H. Orland. Some physical approaches to protein folding. *J. Phys. I France*, 3:259–275, February 1993.

[4] James U. Bowie and David Eisenberg. An evolutionary approach to folding small alpha-helical proteins that uses sequence information and an empirical guiding fitness function. *Proc. Natl. Acad. Sci. USA*, 91:4436–4440, May 1994.

[5] James U. Bowie, Roland Luthy, and David Eisenberg. A method to identify protein sequences that fold into a known three-dimensional structure. *Science*, 253:164–170, July 1991.

[6] Carl Branden and John Tooze. *Introduction to Protein Structure*, pages 4–7. Garland Publishing, Inc., New York and London, 1991.

[7] Kuo-Chen Chou and Chun-Ting Zhang. A correlation-coefficient method to predicting protein-structural classes from amino acid compositions. *Eur. J. Biochem.*, 207:429–433, 1992.

[8] Kuo-Chen Chou and Chun-Ting Zhang. Predicting protein folding types by distance functions that make allowances for amino acid interactions. *The Journal of Biological Chemistry*, 269(35):22014–22020, February 1994.

[9] Kuo-Chen Chou and Chun-Ting Zhang. Prediction of protein structural classes. *Critical Reviews in Biochemistry and Molecular Biology*, 30(4):275–349, 1995.

[10] J.L. Cornette, K.B. Cease, H. Margalit, J.L. Spouge, J.A. Berzofsky, and C. DeLisi. Hydrophobicity scales and computational techniques for detecting amphipathic structures in proteins. *J. Mol. Biol.*, 195:659–685, 1987.

[11] T.E. Creighton. *Proteins: Structures and Molecular Properties*, pages 4,142,202–217. W.H. Freeman, New York, NY, 1993.

[12] Russell F. Doolitle, editor. *Molecular evolution : computer analysis of protein and nucleic acid sequences*, volume 183 of *Methods in Enzymology*, section Profile analysis, pages 146–159. Academic Press, San Diego, California, 1990.

[13] Inna Dubchak, Stephen R. Holbrook, and Sung-Hou Kim. Prediction of protein folding class from amino acid composition. *Proteins: Structure, Function and Genetics*, 16:79–91, 1993.

[14] David Eisenberg, James U. Bowie, Roland Luthy, and Seunghyon Choe. Three-dimensional profiles for analysing protein sequence-structure relationships. *Faraday Discuss.*, 93:25–34, 1992.

[15] Frank Eisenhaber, Cornelius Frommel, and Patrick Argos. Prediction of secondary structural content of proteins from their amino acid composition alone. II. The paradox with secondary structural class. *Proteins: Structure, Function and Genetics*, 25:169–179, 1996.

[16] Michael Gribskov, Andrew D. McLachlan, and David Eisenberg. Profile analysis: Detection of distantly reltated proteins. *Proc. Natl. Acad. Sci. USA*, 84:4355–4358, July 1987.

[17] Teresa Head-Gordon and Frank H. Stillinger. Optimal neural networks for protein-structure prediction. *Physical Review E*, 48(2):1502–1515, August 1993.

[18] George Johnson. Designing life: Proteins 1, computer 0. *The New York Times*, 146(50742), March 1997.

[19] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjolander, and David Haussler. Hidden Markov models in computational biology. *J. Mol. Biol.*, 235:1501–1531, 1994.

[20] Roland Luthy, Ioannis Xenarios, and Philipp Bucher. Improving the sensitivity of the sequence profile method. *Protein Science*, 3:139–146, 1994.

[21] Thomas Madej and Michael C. Mossing. Hamiltonians for protein tertiary structural prediction based on three-dimensional environment principles. *J. Mol. Biol.*, 233:480–487, 1993.

[22] Ken Nishikawa, Yasuchi Kubota, and Tatsuo Ooi. Classification of proteins into groups based on amino acid composition and other characters. II. Grouping into four types. *J. Biochem.*, 94(3):997–1007, 1983.

[23] John Overington, Dan Donnelly, Mark S. Johnson, Angrej Sali, and Tom L. Blundell. Environment-specific amino acid substitution tables: Tertiary templates and prediction of protein folds. *Protein Science*, 1:216–226, 1992.

[24] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[25] Jerry E. Solomon and David Liney. Exploration of compact protein conformations using the guided replication monte carlo method. *Biopolymers*, 36:579–597, 1995.

[26] Collin M. Stultz, James V. White, and Temple F. Smith. Structural analysis based on state-space modeling. *Protein Science*, 2:305–314, 1993.

[27] Gunnar von Heijne. *Sequence Analysis in Molecular Biology*, page 135. Academic Press, Inc., San Diego, CA, 1987.

[28] James V. White, Collin M. Stultz, and Temple F. Smith. Protein classification by stochastic modeling and optimal filtering of amino-acid sequences. *Mathematical Biosciences*, 119:35–75, 1994.

[29] Matthias Wilmanns and David Eisenberg. Inverse protein folding by the residue pair preference profile method: estimating the correctness of alignments of structurally compatible sequences. *Protein Engineering*, 8(7):627–639, 1995.

[30] Kaizhi Yue and Ken A. Dill. Sequence-structure relationships in proteins and copolymers. *Physical Review E*, 48(3):2267–2278, September 1993.

[31] Kam Y.J. Zhang and David Eisenberg. The three-dimensional profile method using residue preference as a continuous function of residue environment. *Protein Science*, 3:687–695, 1994.