Predicting Secondary Structure of All-Helical Proteins Using Hidden Markov Support Vector Machines

Blaise Gassend, Charles W. O'Donnell, William Thies,
Andrew Lee, Marten van Dijk, and Srinivas Devadas

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Contact email: gassend@mit.edu

Abstract. Our goal is to develop a state-of-the-art secondary structure predictor with an intuitive and biophysically-motivated energy model through the use of Hidden Markov Support Vector Machines (HM-SVMs), a recent innovation in the field of machine learning. We focus on the prediction of alpha helices and show that by using HM-SVMs, a simple 7-state HMM with 302 parameters can achieve a Q_α value of 77.6% and a SOV_α value of 73.4%. As detailed in an accompanying technical report [11], these performance numbers are among the best for techniques that do not rely on external databases (such as multiple sequence alignments).

1 Introduction

It remains an important and relevant problem to accurately predict the secondary structure of proteins based on their amino acid sequence. The identification of basic secondary structure elements—alpha helices, beta strands, and coils—is a critical prerequisite for many tertiary structure predictors, which consider the complete three-dimensional protein structure. To date, there has been a broad array of approaches to secondary structure prediction, including statistical techniques, neural networks, Hidden Markov Models, Support Vector Machines, nearest neighbor methods and energy minimization. In terms of prediction accuracy, neural networks are among the most popular methods in use today [10,14], delivering a pointwise prediction accuracy (Q_3) of about 77% and a segment overlap measure (SOV) [19] of about 74% [8].

However, to improve the long-term performance of secondary structure prediction, it likely will be necessary to develop a cost model that mirrors the underlying biological constraints. While neural networks offer good performance today, their operation is largely opaque. Often containing up to 10,000 parameters and relying on complex layers of non-linear perceptrons, neural networks offer little insight into the patterns learned. Moreover, they mask the shortcomings of the underlying models, rendering it a tedious and ad-hoc process to improve them. In fact, over the past 15 years, the largest improvements in

neural network prediction accuracy have been due to the integration of homologous sequence alignments [8] rather than specific changes to the underlying cost model.

Of the approaches developed to date, Hidden Markov Models (HMMs) offer perhaps the most natural representation of protein secondary structure. An HMM consists of a finite set of states with learned transition probabilities between states. In biological terms, each transition corresponds to a local folding event, with the most likely sequence of states corresponding to the lowest-energy protein structure. HMMs generally contain hundreds of parameters, 1-2 orders of magnitude less than that of neural networks. In addition to providing a tractable model that can be reasoned about, the reduction in parameters lessens the risk of overlearning. However, the leading HMM methods to date [5,18] have not exceeded a Q_3 value of 75%, and SOV scores are often unreported.

In this paper, we focus on improving the prediction accuracy of HMM-based methods, thereby advancing the goal of achieving a state-of-the-art predictor while maintaining an intuitive and biophysically-motivated cost model. Our technique relies on Hidden Markov SVMs (HM-SVMs), a recent innovation in the field of machine learning [1]. While HM-SVMs share the prediction structure of HMMs, the learning algorithm is more powerful. Unlike the expectation-maximization algorithms typically used to train HMMs, training with an SVM allows for a discriminative learning function, a soft margin criterion, and bi-directional influence of features on parameters [1].

Using the HM-SVM approach, we develop a simple 7-state HMM for predicting alpha helices and coils. The HMM contains 302 parameters, representing the energetic benefit for each residue being in the middle of a helix or being in a specific position relative to the N- or C-cap. Our technique does not depend on any homologous sequence alignments. Applied to a database of all-alpha proteins, our predictor achieves a Q_α value of 77.6% and an SOV_α score of 73.4%. Among other HMMs that do not utilize alignment information, it appears that our Q_α represents a 3.5% improvement over the previous best [12], while our SOV_α is comparable (0.2% better). However, due to differences in the data set, we emphasize the novelty of the approach rather than the exact magnitude of the improvements. We are extending our technique to beta strands (and associated data sets) as ongoing work.

2 Algorithm

2.1 Formal Optimization Problem

It is widely believed that when a protein is folded, its free-energy approaches a thermodynamic minimum. In our technique, we define a free-energy function $G(\mathbf{x}, \mathbf{y})$ that estimates the free-energy of an amino acid sequence $\mathbf{x}$ when folded into a candidate secondary structure $\mathbf{y}$. Our predictor outputs the secondary structure $\hat{\mathbf{y}}$ that has the minimal free-energy according to G:

$$\hat{\mathbf{y}} = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}} G(\mathbf{x}, \mathbf{y}). \tag{1}$$

To go from this general statement to a working algorithm, we need to a find free-energy function G and a set of structures $\mathcal{Y}$ for which the minimization shown in equation (1) is easy to compute. In choosing G and $\mathcal{Y}$, we tradeoff the ability to efficiently minimize G with the ability to accurately capture the richness and detailed physics of protein structure. Atomistic models are able to capture the whole range of structures, and incorporate all the physical interactions between atoms. However, because of this detail they can only be optimized using heuristic methods. We therefore prefer to consider a simplified set of structures $\mathcal{Y}$, and a cost function G with lumped parameters that try to approach physical reality.

These lumped parameters are difficult to determine experimentally. We will therefore define a class $\mathcal{G}$ of candidate free-energy functions that are easy to optimize over some set of structures $\mathcal{Y}$. Then we will use machine learning techniques to pick a good G from all the candidates in $\mathcal{G}$. The machine learning will use structure information from the Protein Data Bank (PDB) [4] to determine which G to pick. Given a set of training examples $\{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \ldots, k\}$, the learning algorithm needs to find a $G \in \mathcal{G}$ such that:

$$\forall i : \mathbf{y}_i = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}} G(\mathbf{x}_i, \mathbf{y}). \tag{2}$$

In practice, this G may not exist or may not be unique, so the machine learning algorithm may have to pick a good approximation, or select a G that is more likely to generalize well to proteins not in the training set. We will now look more closely at how a good G is selected, and later, in Section 2.5, we will be more specific about what $\mathcal{G}$ and $\mathcal{Y}$ are.

2.2 Iterative Constraint Based Approach

First, we notice that equation (2) can be rewritten as the problem of finding a function G that satisfies the large set of inequality constraints

$$\forall i, \forall y \in \mathcal{Y} \setminus \{\mathbf{y}_i\} : G(\mathbf{x}_i, \mathbf{y}_i) < G(\mathbf{x}_i, \mathbf{y}). \tag{3}$$

Unfortunately, the set of all secondary structures $\mathcal{Y}$ is exponentially large, so finding a $G \in \mathcal{G}$ that satisfies all these inequalities directly is computationally intractable. Our approach reduces the problem by ignoring as many constraints as possible, only considering the constraints it is "forced" to consider.

In our method, the reduced problem is defined as the problem of finding a function G' that satisfies the set of constraints

$$\forall i, \forall y \in S_i : G'(\mathbf{x}_i, \mathbf{y}_i) < G'(\mathbf{x}_i, \mathbf{y}), \tag{4}$$

for some $S_i \subseteq \mathcal{Y} \setminus \{\mathbf{y}_i\}$.

Initially, we begin with no constraints at all (that is, $S_i = \emptyset$ for all i) and we choose some function $G' \in \mathcal{G}$. Note that, since we start with no constraints, any function $G' \in \mathcal{G}$ initially satisfies equation (4). We then need to check whether G'

approximates the solution G to the set of constraints (2). In particular, we verify whether G' can be used to approximate $\mathbf{y}_1$ as the solution $\hat{\mathbf{y}}_1$ of the problem

$$\hat{\mathbf{y}}_1 = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}} G'(\mathbf{x}_1, \mathbf{y}).$$

If $G'(\mathbf{x}_1, \mathbf{y}_1) < G'(\mathbf{x}_1, \hat{\mathbf{y}}_1) + \varepsilon$, we say that $\hat{\mathbf{y}}_1$ is "close" to $\mathbf{y}_1$ in the sense that $\hat{\mathbf{y}}_1$ is a close enough approximation of $\mathbf{y}_1$. If $\hat{\mathbf{y}}_1$ is close to $\mathbf{y}_1$, we go on to the next optimization problem,

$$\hat{\mathbf{y}}_2 = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}} G'(\mathbf{x}_2, \mathbf{y}).$$

If $\hat{\mathbf{y}}_1$ is not close to $\mathbf{y}_1$, this means the constraint $G'(\mathbf{x}_1, \mathbf{y}_1) < G'(\mathbf{x}_1, \hat{\mathbf{y}}_1)$ in equation (3) has been violated. Therefore we must add this constraint to our reduced problem, replacing S_1 by $S_1 \cup \{\hat{\mathbf{y}}_1\}$. In order to solve the new reduced problem we need to find a new G' that satisfies the old and new constraints. At all times the number of constraints in the reduced problem is relatively small such that it is computationally feasible to find its solution.

Whenever a prediction $\hat{\mathbf{y}}_i$ is not satisfactorily close to $\mathbf{y}_i$, we add more constraints. For instance, Figure 1 shows our problem reduction for the training example $(\mathbf{x}_1, \mathbf{y}_1)$. Note that the reduced problems lead to the constraints $G'(\mathbf{x}_1, \mathbf{y}_1) < G'(\mathbf{x}_1, \mathbf{y}^1)$, $G'(\mathbf{x}_1, \mathbf{y}_1) < G'(\mathbf{x}_1, \mathbf{y}^7)$, $G'(\mathbf{x}_1, \mathbf{y}_1) < G'(\mathbf{x}_1, \mathbf{y}^{245})$, etc., where $\mathcal{Y} = \{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^m\}$ (in other words, $S_1 = \{\mathbf{y}^1, \mathbf{y}^7, \mathbf{y}^{245}\}$).

The algorithm terminates if no constraints need to be added. That is, each prediction is a good approximation,

$$\forall i : G'(\mathbf{x}_i, \mathbf{y}_i) < G'(\mathbf{x}_i, \hat{\mathbf{y}}_i) + \varepsilon \text{ where } \hat{\mathbf{y}}_i = \operatorname*{argmin}_{\mathbf{y} \in \mathcal{Y}} G'(\mathbf{x}_i, \mathbf{y}). \tag{5}$$

This is equivalent to

$$\forall i, \forall y \in \mathcal{Y} \setminus \{\mathbf{y}_i\} : G'(\mathbf{x}_i, \mathbf{y}_i) < G'(\mathbf{x}_i, \mathbf{y}) + \varepsilon. \tag{6}$$

This is similar to the full set of constraints on G in equation (3), except that G' need only satisfy each inequality within a distance of ε.

2.3 Linear Cost Function

One important assumption we make is that the family of free energy functions $\mathcal{G}$ is linear. That is, the total free energy of the protein is a sum of elementary interactions. This simplification agrees with many mathematical models of the energy force fields that control protein folding. For example, electrostatic, Van der Waals, stretch, bend, and torsion forces can all be described by the sum of energy terms for each pair of molecular elements. Given this, we can formally define the family of functions $\mathcal{G}$ to be

$$\mathcal{G} = \{G_{\mathbf{w}} : (\mathbf{x}, \mathbf{y}) \longrightarrow \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle : \text{ for some } \mathbf{w}\}. \tag{7}$$

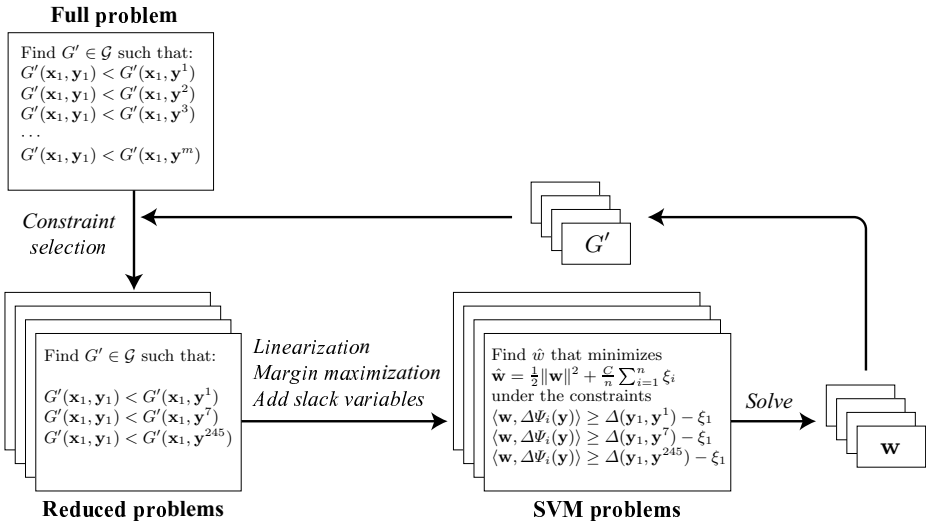

Fig. 1. Summary of the learning method. In this figure each large frame represents a problem that needs to be solved. On the left, we start with an intractably large problem. At each iteration, we pick a subset of the large problem to work on, solve it approximately using an SVM formulation, and use the resulting solution to expand the subset of constraints we are working with.

Here the feature function Ψ is fixed and known, representing the specific energy characteristics that we are interested in. For example, one element of the vector $\Psi(\mathbf{x}, \mathbf{y})$ might be the number of proline residues from sequence $\mathbf{x}$ that appear within an alpha helix in candidate structure $\mathbf{y}$. Additional details on our design of Ψ appear in Sections 2.5 and 3.1. By definition of a linear function, the dot product of the vector $\mathbf{w}$ (notated by $\langle , \rangle$) can then be taken to appropriately weight the importance of individual terms within Ψ. With this assumption, the reduced problem's constraints given by equation (4) can be rewritten as

$$\forall i, \forall y \in S_i : G_{\mathbf{w}}(\mathbf{x}_i, \mathbf{y}_i) < G_{\mathbf{w}}(\mathbf{x}_i, \mathbf{y}). \tag{8}$$

In order to solve the reduced problem, we need to find the unknown weight vector $\mathbf{w}$ such that these constraints are satisfied. Again, since $G_{\mathbf{w}}$ is a linear function, this set of constraints can translate into

$$\forall i, \forall \mathbf{y} \in S_i : \langle \mathbf{w}, \Delta\Psi_i(\mathbf{y}) \rangle > 0, \tag{9}$$

where $\Delta\Psi_i(\mathbf{y}) = \Psi(\mathbf{x}_i, \mathbf{y}) - \Psi(\mathbf{x}_i, \mathbf{y}_i)$. This reformulation of the constraints allows this problem to be solved in a much more elegant and computationally efficient manner. We use the powerful technique of Support Vector Machines to quickly determine the function $G_{\mathbf{w}}$, although many other techniques are possible.

2.4 Iteratively Constraining Support Vector Machines

Support Vector Machines (SVMs) are a fast and effective tool for generating functions from a set of labeled input training data. SVMs are able to determine a set of weights $\mathbf{w}$ for the function $G_{\mathbf{w}}$ that will allow $G_{\mathbf{w}}$ to accurately map all of the training example inputs $\mathbf{x}_i$ to outputs $\mathbf{y}_i$. This problem can be formulated as a quadratic program, in which the variables are the weights $\mathbf{w}$ and a set of "slack variables" ξ_i:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \tag{10a}$$

under the constraints

$$\forall i, \forall \mathbf{y} \in S_i : \langle \mathbf{w}, \Delta\Psi_i(\mathbf{y}) \rangle \geq 1 - \xi_i \quad \text{with} \quad \forall i : \xi_i \geq 0. \tag{10b}$$

The only differences between these constraints and those in equation (9) is that *(i)* the strict inequality (> 0) is replaced by a non-strict inequality (≥ 1), and *(ii)* slack variables ξ_i are introduced to allow a best-fit solution in the event of unsatisfiable constraints. The objective function minimizes the length of the weight vector (to normalize the constraints across various dimensions of $\mathbf{w}$) and the size of the slack variables. The constant parameter C indicates how much a solution is penalized for violating a constraint. In practice, SVMs solve the dual of the this minimization problem.

We can therefore use SVMs to determine our function $G_{\mathbf{w}}$; however, this only solves half of our problem. Given a candidate $G_{\mathbf{w}}$ we must then determine if equation (3) has been violated and add more constraints to it if necessary. To accomplish this task, we build off of work done by Tsochantaridis et al. [17] which tightly couples this constraint verification problem with the SVM $\mathbf{w}$ minimization problem.

First a loss function $\Delta(\mathbf{y}_i, \mathbf{y})$ is defined that weighs the goodness of the structures $\hat{\mathbf{y}}_i$. Smaller values of $\Delta(\mathbf{y}_i, \mathbf{y})$ indicate that structures $\mathbf{y}_i$ and $\mathbf{y}$ are more similar; see Section 3.1 for examples. Adding this to the SVM constraints in equation (10b) gives

$$\forall i, \forall \mathbf{y} \in S_i : \xi_i \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \langle \mathbf{w}, \Delta\Psi_i(\mathbf{y}) \rangle. \tag{11}$$

Using this we can decide when to add constraints to our reduced problem and which constraints to add. Since at every iteration of the algorithm we determine some $\mathbf{w}$ for the current S_i, we can then find the value $\hat{\xi}_i$ assigned to variable ξ_i as a result of the optimization. $\hat{\xi}_i$ corresponds to the "worst" prediction by $\mathbf{w}$ across the structures $\mathbf{y} \in S_i$:

$$\hat{\xi}_i = \max(0, \max_{\mathbf{y} \in S_i} \Delta(\mathbf{y}_i, \mathbf{y}) - \langle \mathbf{w}, \Delta\Psi_i(\mathbf{y}) \rangle). \tag{12}$$

This resulting $\hat{\xi}_i$, which was determined using S_i, can be compared to a similar $\hat{\xi}_i'$ that is obtained by instead maximizing over $\mathcal{Y} \setminus \{\mathbf{y}_i\}$ in equation (12).

```
Input: (x_1, y_1), ..., (x_n, y_n), C, ε
S_i ← ∅ for all 1 ≤ i ≤ n
w ← any arbitrary value
repeat (
 for i = 1, ..., n do (
  Set up the cost function:
   H(y) = Δ(y_i, y) − ⟨w, ΔΨ_i(y)⟩
  Compute ŷ = argmax_{y∈Y\{y_i}} H(y)
  Compute ξ̂_i = max{0, max_{y∈S_i} H(y)}
  if H(ŷ) > ξ̂_i + ε then (
   S_i ← s_i ∪ {ŷ}
   w ← optimize over S = ∪_i S_i
))) until no S_i changes during iteration
```

Algorithm 1. Algorithm for iterative constraint based optimization.

This will tell us how much the constraints we are ignoring from $\mathcal{Y} \setminus \{\mathbf{y}_i\}$ will change the solution. The constraint that is most likely to change the solution is that which would have caused the greatest change to the slack variables. Therefore we would add the constraint to S_i that corresponds to

$$\hat{\mathbf{y}}' = \underset{\mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}_i\}}{\operatorname{argmax}} \Delta(\mathbf{y}_i, \mathbf{y}) - \langle \mathbf{w}, \Delta\Psi_i(\mathbf{y}) \rangle. \tag{13}$$

Tsochantaridis et al. [17] show that by only adding constraints when $\hat{\mathbf{y}}'$ could change $\hat{\xi}_i$ by more than ε, one can attain a provable termination condition for the problem. The summary of this overall process appears in Algorithm 1.

2.5 Defining the Set of Valid Structures

One final issue remains to be solved to complete our algorithm. We need to specify what $\mathcal{Y}$ and $\Psi(\mathbf{x}, \mathbf{y})$ are, and how to optimize $G(\mathbf{x}, \mathbf{y})$ over $\mathcal{Y}$. In general, $\mathcal{Y}$ can be exponentially large with respect to the sequence length, making brute-force optimization impractical. Our general approach is to structure $\mathcal{Y}$ and $\Psi(\mathbf{x}, \mathbf{y})$ in a way that allows optimization of $G(\mathbf{x}, \mathbf{y})$ through dynamic programming.

Most secondary-structure prediction tools use local features to predict which regions of a protein will be helical [14]. Individual residues can have propensities for being in a helix, they can act as helix nucleation sites, or they can interact with other nearby residues. This type of information can be well captured by Hidden Markov Models (HMMs). Equivalently, we choose to capture them using Finite State Machines (FSMs). The only difference between the FSMs we use and a non-stationary HMM is that the HMM deals with probabilities, which are multiplicative, while our FSMs deal with pseudo-energies, which are additive. To a logarithm, they are the same.

We define $\mathcal{Y}$ to be the language that is recognized by some FSM. Thus a structure $\mathbf{y} \in \mathcal{Y}$ will be a string over the input alphabet of the FSM. For

example, that alphabet could be $\{h, c\}$, where h indicates that the residue at that position in the string is in a helix, and c indicates that it is in a coil region. A string $\mathbf{y}$ is read by an FSM one character at a time, inducing a specific set of transitions between internal states. Note that the FSMs we are considering do not need to be deterministic. However, they do need to satisfy the property that, for a given input string, there is at most one set of transitions leading from the initial state to a final state. We denote this sequence of transitions by $\sigma(\mathbf{y})$ and note that $\sigma(\mathbf{y})$ need not be defined for all $\mathbf{y}$.

To define $\Psi(\mathbf{x}, \mathbf{y})$, we create a helper function $\psi(\mathbf{x}, t, i)$ which assigns a vector of feature values whenever transition t is taken at position i in the sequence $\mathbf{x}$. For example, if a transition is taken to start a helix at position i, then $\psi(\mathbf{x}, t, i)$ might return features indicating that residues at position $i - 3$ to $i + 3$ are associated with an N-terminal helix cap. See Section 3.1 for our particular choice of ψ. The overall feature vector is the sum of these features across all positions in the sequence: $\Psi(\mathbf{x}, \mathbf{y}) = \sum_i \psi(\mathbf{x}, \sigma(\mathbf{y})_i, i)$.

The total cost $G(\mathbf{x}, \mathbf{y})$ follows the form of equation (7). We also specify an infinite cost for structures that are the wrong length or are rejected by the FSM:

$$G(\mathbf{x}, \mathbf{y}) = \begin{cases} +\infty & \text{if } |\mathbf{x}| \neq |\mathbf{y}| \text{ or } \sigma(\mathbf{y}) \text{ is undefined} \\ \langle w, \Psi(\mathbf{x}, \mathbf{y}) \rangle & \text{otherwise} \end{cases} \tag{14}$$

This cost is easy to optimize over $\mathcal{Y}$ by using the Viterbi algorithm. This algorithm proceeds in $|\mathbf{x}|$ rounds. In round i, the best path of length s starting from an initial state is calculated for each FSM state. These paths are computed by extending the best paths from the previous round by one transition, and picking the best resulting path for each state. The algorithmic complexity is $O(|FSM| \cdot |\mathbf{x}|)$, where $|FSM|$ is the number of states and transitions in the FSM.

3 Results

We now present results from our implementation of our algorithm. It was written in Objective Caml, and uses SVM^{struct}/SVM^{light} [7] by Thorsten Joachims.

3.1 Finite State Machine Definition

In our experiments, we have used an extremely simple finite state machine that is presented in Figure 2. Each state corresponds to being in a helix or coil region, and indicates how far into the region we are. States `H4` and `C3` correspond to helices and coils more than 4 and 3 residues long, respectively. Short coils are permitted, but helices shorter than 4 residues are not allowed, as even 3_{10} helices need at least 4 residues to complete one turn and form the first hydrogen bond.

Table 1 lists the basic features that were used in our experiments. These features can also be considered to be the parameters of our system, as our learning algorithm assigns an appropriate weight to each one. Our choice of features is motivated by observations that amino acids have varying propensities for appearing within an alpha helix as well as for appearing at the ends of a helix,

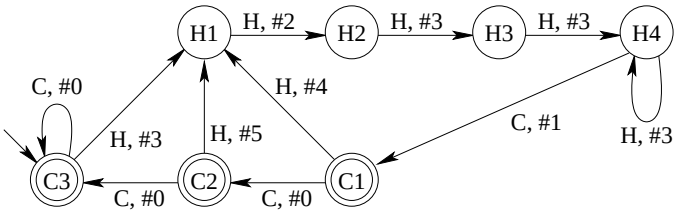

Fig. 2. The finite state machine we used. Double circles represent accept states. The arrow leading into state `C3` indicates that it is an initial state. Each transition is labeled with the type of structure it corresponds to: helix (`H`) or coil (`C`), and a label (`#i`) indicating which features correspond to this transition in Table 2.

Name	**Number of features**	**Description**
A	1	Penalty for very short coil
B	1	Penalty for short coil
H_R	20	Energy of residue R in a helix
C_R^i	140	Energy of residue R at position i relative to C-cap
N_R^i	140	Energy of residue R at position i relative to N-cap
Total	302	

Table 1. Summary of basic features that are considered. Each of these features corresponds to a parameter that is learned by our algorithm.

Label	**Features**	**Description**
#0	0	Coil defined as zero-energy
#1	$\sum_{i=-3}^{+3} C_{R_{n+i-1}}^{i-1}$	End of helix processing (C-cap)
#2	$H_{R_n} + \sum_{i=-3}^{+3} N_{R_{n+i-1}}^{i-1}$	Start of helix processing (N-cap)
#3	H_{R_n}	Normal helix residue
#4	$H_{R_n} + A$	Helix after very short coil
#5	$H_{R_n} + B$	Helix after short coil

Table 2. Sets of features that are emitted by transitions in the FSM. R_i denotes the residue at position i in the protein, and n is the current position of the FSM.

an area termed the helix cap [2]. We introduce a single feature per residue to account for helix propensity, for a total of 20 parameters. For helix capping, we use a separate feature for each residue that appears at a given offset (-3 to $+3$) from a given end of the helix (N-terminal or C-terminal). This accounts for $20 * 7 * 2 = 280$ parameters. Finally, we also introduce a feature for very short (2-residue) and short (3-residue) coils. Thus, there are a total of 302 parameters.

Table 2 illustrates how features are associated with the transitions of the FSM. This table corresponds to the ψ function described in Section 2.5; given an FSM transition and a position in the input sequence, it outputs a set of representative features. Most of this mapping is straightforward. In the case of

Description	SOV_α (%) (train)	SOV_α (%) (test)	Q_α (%) (train)	Q_α (%) (test)	Training time (s)
Best run for SOV_α	76.4	75.1	79.6	78.6	123
Average of 20 runs	75.1	73.4	79.1	77.6	162
Standard deviation of 20 runs	1.0	1.4	0.6	0.9	30

Table 3. Results of our predictor across 20 configurations of training/test set.

helix caps (labels #1 and #2), features are emitted across a 7-residue window that is centered at position $n-1$ (the previously processed residue).

None of the features we have used involve more than one residue in the sequence. We have experimented with more complicated cost functions that model pairwise interactions between nearby residues in a helix, namely between n and $n+3$ or n and $n+4$. So far we have not managed to improve our prediction accuracy using these interactions, possibly because each pairwise interaction adds 400 features to the cost function, leaving much room for over-learning. Indeed, with the expanded cost functions we observed improved predictions on the training proteins, but decreased performance on the test proteins.

We have also experimented with various loss functions Δ (see Section 2.4). We have tried a 0-1 loss function (0 unless both structures are identical), hamming distance (number of incorrectly predicted residues), and a modified hamming distance (residues are given more weight when they are farther from the helix-coil transitions). Each one gives results slightly better than the previous one.

3.2 Results

We have been working with a set of 300 non-homologous all-alpha proteins taken from EVA's largest sequence-unique subset [6] of the PDB [4] at the end of July 2005. The sequences and structures have been extracted from PDB data processed by DSSP [9]. Only alpha helices have been considered (`H` residues in DSSP files); everything else has been lumped as *coil* regions.

In our experiments, we split our 300 proteins into two 150 protein subsets. The first set is used to train the cost function; the second set is used to evaluate the cost function once it has been learned. Since the results vary a bit depending on how the proteins are split in two sets, we train the cost function on 20 random partitions into training and test sets, and report the average performance.

Table 3 presents our results using both the Q_α and SOV_α metrics. The Q_α metric is simply the number of correctly predicted residues divided by sequence length. SOV_α is a more elaborate metric that has been designed to ignore small errors in helix-coil transition position, but heavily penalize more fundamental errors such as gaps appearing in a helix [19].

The weights obtained for the features in Table 1 are available in our technical report [11] (although their sign is reversed relative to this paper). Initial examination has shown some correlation with propensities found in the literature [2].

Our experiments utilized a slack variable weighting factor $C = 0.08$ in equation (10a). The algorithm termination criterion was for $\varepsilon = 0.1$. Both of these

parameters have a large impact on prediction accuracy and training time. Our choice of these values was driven by informal experiments in which we tried to maximize the test accuracy while maintaining a practical training time.

4 Related Work

Tsochantaridis et al. apply an integrated HMM and SVM framework for secondary structure prediction [16]. The technique may be similar to ours, as we are reusing their SVM code; unfortunately, there are few details published.

Though state-of-the-art neural network predictors such as PSIPred [8] currently out-perform our method by about 5%, they incorporate multiple sequence alignments and are often impervious to analysis and understanding. For example, the PHD predictor contains more than 10,000 parameters [15], and SSPro contains between 1,400 and 2,900 parameters [3]. A notable exception is the network of Riis and Krogh [13], which is structured by hand to reduce the parameter count to as low as 311 (prediction accuracy is reported at $Q_3 = 71.3\%$). In comparison, our technique uses 302 parameters and offers $Q_3 = 77.6\%$. Also, we do not incorporate alignment information, which is often responsible for 5-7% improvement in accuracy [13,15].

Please see our technical report for a complete discussion of related work [11].

5 Conclusion

In this paper, we present a method to predict alpha helices in all-alpha proteins. The HMM is trained using a Support Vector Machine method which iteratively picks a cost function based on a set of constraints, and uses the predictions resulting from this cost function to generate new constraints for the next iteration.

On average, our method is able to predict all-alpha helices with an accuracy of 73.4% (SOV_α) or 77.6% (Q_α). Unfortunately, these results are difficult to compare with existing prediction methods which usually do predictions on both alpha helices and beta strands. Rost and Sanders caution that restricting the test set to all-alpha proteins can result in up to a 3% gain in accuracy [15]. In addition, recent techniques such as PSIPred [8] consider 3_{10} helices (the DSSP state 'G') to be part of a helix rather than loop, and report gains of about 2% in overall Q_3 if helices are restricted to 4-helices (as in most HMM techniques, including ours).

The real power of the machine learning method we use is its applicability beyond HMM models. Instead of describing a protein structure as a sequence of HMM states, we could equally describe it as a parse tree of a context-free grammar or multi-tape grammar. With these enriched descriptions, we should be able to include in the cost function interactions between adjacent strands of a beta sheet. This should allow us to incorporate beta sheet prediction into our algorithm. Unlike most secondary structure methods, we would then be able to predict not only which residues participate in a beta sheet, but also which residues are forming hydrogen bonds between adjacent sheets.

6 Acknowledgements

We thank Chris Batten, Edward Suh and Rodric Rabbah for their early contributions to this work, and the anonymous reviewers for their helpful comments. W.T. also thanks Saman Amarasinghe for supporting his part in this research.

References

1. Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden Markov Support Vector Machines. In *ICML*, 2003.
2. R. Aurora and G. Rose. Helix capping. *Protein Science*, 7, 1998.
3. P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15, 1999.
4. H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, 28, 2000.
5. C. Bystroff, V. Thorsson, and D. Baker. HMMSTR: a Hidden Markov Model for Local Sequence-Structure Correlations in Proteins. *J. of Mol. Bio.*, 301, 2000.
6. EVA Largest sequence of unique subset of PDB. `http://salilab.org/~eva/res/weeks.html#unique`.
7. T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods – Support Vector Learning*, pages 169–185. MIT Press, 1998.
8. D. T. Jones. Protein Secondary Structure Prediction Based on Position-specific Scoring Matrices. *Journal of Molecular Biology*, 292:195–202, 1999.
9. W. Kabsch and C. Sander. Dictionary of protein secondary structure. *Biopolymers*, 22, 1983.
10. V. Eyrich et al. EVA: Continuous automatic evaluation of protein structure prediction servers. *Bioinformatics*, 17(12):1242–1243, 2001.
11. B. Gassend et al. Secondary Structure Prediction of All-Helical Proteins Using Hidden Markov Support Vector Machines. Technical Report MIT-CSAIL-TR-2005-060, MIT, December 2005. `http://hdl.handle.net/1721.1/30571`.
12. M. N. Nguyen and J. C. Rajapakse. Prediction of protein secondary structure using bayesian method and support vector machines. In *ICONIP*, 2002.
13. S. Riis and A. Krogh. Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments. *Journal of Computational Biology*, 3:163–183, 1996.
14. B. Rost. Review: Protein Secondary Structure Prediction Continues to Rise. *Journal of Structural Biology*, 134(2):204–218, 2001.
15. B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, 232:584–599, 1993.
16. I. Tsochantaridis, Y. Altun, and T. Hoffman. A crossover between SVMs and HMMs for protein structure prediction. In *NIPS Workshop on Machine Learning Techniques for Bioinformatics*, 2002.
17. I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support Vector Machine Learning for Interdependent and Structured Output Spaces. In *ICML*, 2004.
18. K. Won, T. Hamelryck, A. Prügel-Bennett, and A. Krogh. Evolving Hidden Markov Models for Protein Secondary Structure Prediction. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 33–40, 2005.
19. A. Zemla, Česlovas Venclovas, K. Fidelis, and B. Rost. A Modified Definition of Sov, a Segment-Based Measure for Protein Secondary Structure Prediction Assessment. *Proteins*, 34(2):220–223, 1999.