SINGLE-SEQUENCE PROTEIN SECONDARY STRUCTURE PREDICTION BY NEAREST-NEIGHBOR CLASSIFICATION OF PROTEIN WORDS

SINGLE-SEQUENCE PROTEIN SECONDARY STRUCTURE PREDICTION BY NEAREST-NEIGHBOR CLASSIFICATION OF PROTEIN WORDS

By

DAVID JONATHAN PORFIRIO

A Thesis Submitted to The Honors College

In Partial Fulfillment of the Bachelors Degree
With Honors In

Computer Science

THE UNIVERSITY OF ARIZONA

MAY 2016

Approved by:

_____

Dr. John Kececioglu
Department of Computer Science

Single-Sequence Protein Secondary Structure Prediction by Nearest-Neighbor Classification of Protein Words

David Porfirio

Abstract

Predicting protein secondary structure is the process by which, given a sequence of amino acids as input, the secondary structure class of each position in the sequence is predicted. Our approach is built on the extraction of protein words of a fixed length from protein sequences, followed by nearest-neighbor classification in order to predict the secondary structure class of the middle position in each word. We present a new formulation for learning a distance function on protein words based on position-dependent substitution scores on amino acids. These substitution scores are learned by solving a large linear programming problem on examples of words with known secondary structures. We evaluated this approach by using a database of 5519 proteins with a total amino acid length of approximately 3000000. Presently, a test scheme using words of length 23 achieved a uniform average over word position of 65.2%. The average accuracy for alpha-classified words in the test was 63.1%, for beta-classified words was 56.6%, and for coil classified words was 71.6%.

CONTENTS

1 INTRODUCTION

Proteins are biological polymers composed of a chain of smaller molecules called amino acids. A protein's primary structure describes the sequence of amino acids, whereas its secondary structure describes the local three-dimensional formations that arise in-sequence due to the chemical interactions between amino acids in the chain. There are three general classes of secondary structures – (1) alpha helices, in which amino acids form a helical structure; (2) beta sheets, which which parallel strands of amino acids connect laterally to each other; and (3) coil structures that include formations not classified as alpha helix or beta sheet (Fig. 1).

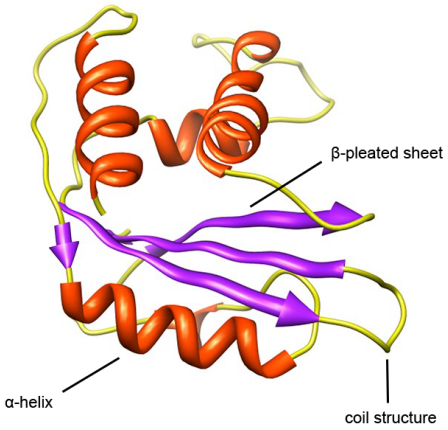

Figure 1: Example protein with all three basic secondary structures (Aslanzadeh, 2012)[1]. Note that the coil structure includes every structure not an alpha helix or beta sheet.

It is possible to predict a protein's secondary structure given its primary structure with high accuracy. PSIPRED is an popular example of software that accomplishes this task. Given an amino acid sequence as input, PSIPRED uses PSI-BLAST to perform a database search for proteins with a similar amino acid sequence, called homologs. PSIPRED then uses neural networks to classify the protein's secondary structure (Jones, 1999)[4]. In contrast, single-sequence prediction only uses the input protein sequence without a database of protein sequences, and does not need to find homologs through costly database searches. This study presents a novel way to predict protein secondary structure via single-sequence prediction on a set of training words with known secondary structures by performing nearest-neighbor classification on words of a fixed length centered on all of the positions of a protein.

Nearest-neighbor classification is a technique that predicts the structure class of a protein word using a database of other protein words labeled by class.

In our implementation, an alpha-distance function compares a query word to the set of all words with known alpha classifications, called the *alpha-touchstone*. A different beta distance function compares the same query word to the set of all words with known beta classification, called the *beta-touchstone*. The secondary structure classifications of the query word's k-nearest neighbors determine whether the query word is classified as alpha or beta, or coil if none of the nearest neighbors are close enough to the query word. Both distance functions may be collectively referred to as the "distance function" for the purposes of this investigation.

Before prediction occurs, a default distance function exists, and a new distance function is learned through linear programming on as set of training words with known secondary structures. The linear program, solved by IBM's CPLEX Optimization Studio, minimizes the degree to which the default distance function mis-classifies the secondary structures of training words[3]. The solution to the linear program is an improved distance function, which replaces the default distance function. Continual improvement of the distance function is possible through setting up another linear program with the improved default distance function as input, and then iterating on the distance function.

The computational approach section of this investigation provides greater detail about prediction and learning the distance function. The experimental results section shows that our current single-sequence computational approach yields improved accuracies on classifying protein words from the single-sequence computational approach in previous work done by Benjamin Yee in John Kececioglu's research group. In the previous work, the accuracies on disjoint training words were 58.0% in the alpha class, 44.1% in the beta class, 81.1% in the coil class, and 65.6% as the position-wise average, which averages accuracies uniformly over all positions in the protein. The class-wise average was 61.1%, which is the average of the accuracies from each individual class. The new computational approach on disjoint training words yields an average of 67.3% for words in the alpha class, 62.6% for words in the beta class, 75.5% for words in the coil class, and 68.5% as the position-wise average. The same number of words were predicted on in each class, so the class-wise average is the same as the position-wise average.

Classifying words in the disjoint testing data, which contained words that were not used to train the distance function, also yielded improved accuracies. The previous computational approach reported 56.3% accuracy for predicting alpha testing words, 41.5% for beta testing words, 79.1% for coil testing words, 60.0% averaged class-wise, and 63.3% averaged position-wise. In contrast, the current computation approach yields accuracies of 63.1% for alpha testing words, 56.6% for beta testing words, 71.6% for coil testing words, 63.8% averaged class-wise, and 65.2% averaged position-wise. Despite these significant improvements, our prediction accuracies are not yet competitive with those of PSIPRED's non-single-sequence approach, which reports average accuracies above 75%[4].

2 COMPUTATIONAL APPROACH

2.1 *The Distance Function*

The distance function between two protein words is fundamental to predicting the secondary structure of a word. Given a query word and another word with known classification, the distance function will compute their likeness, or distance from each other. It does this by calculating the cost of substituting an amino acid at a specific position in one word with the amino acid at the same position in the other word. These costs are called substitution scores, whose values are dependent on the class X of the known word, the position i of the amino acids being substituted, and the amino acids' identities. The notation for a substitution score will therefore be $S_{X,i}(a,b)$ for $X \in$ *classes*, $i \in$ *positions in both words*, and $a, b \in$ *the set of amino acids*. The distance function of a query word and a known word is simply the sum of all of the substitution scores between corresponding amino acids, or

$$d_X(w,v) = \sum_{i \text{ in } w,v} S_{X,i}(a,b) \quad \text{for word } w \in \text{query words and word } v \in X \quad (1)$$

The distance function can therefore be thought of as two separate distance functions, one that computes the distance from a query word to a known alpha word, and another that computes the distance between a query word and known beta word. This is what is meant when the "alpha" or "beta" distance functions are discussed. There is no distance function for coil words, which will be discussed in the next section.

2.2 *Predicting Secondary Structure Given a Distance Function and a Coil/Non-coil Threshold*

The distance of a word with unknown secondary structure to its alpha and beta nearest neighbors determines whether that word is more likely to be classified as alpha or beta. These distances might be significantly large, however. Therefore, there exists a general mechanism by which a word with unknown secondary structure is classified: if the word has significantly large distances to its alpha and beta nearest neighbors, then it is classified as a coil word; else, it is classified as either an alpha or beta word depending on whether the alpha or beta nearest neighbors are closer. A *coil/non-coil threshold* distance, τ, is used to decide whether a word is too far from its alpha and beta nearest neighbors. If a word's distance to it's alpha and beta nearest neighbors is above τ, then the word is classified as coil. Finding a value of τ that optimizes prediction accuracies is detailed in section 2.4.

When given a query word w, the first priority is to classify it as coil or non-coil. In order to do so, one alpha nearest neighbor a and one beta nearest neighbor b must be found using the distance function. There are then three

different schemes, termed *Coil Schemes*, that were implemented in the coil classification process. Each coil scheme involves a different way of calculating the *critical nearest-neighbor distance (CNND)*, or the value that will determine whether the query word is above or below τ:

Min Coil Scheme (default): $CNND = min[d_A(w,a),\ d_B(w,b)]$

Max Coil Scheme 2: $CNND = max[d_A(w,a),\ d_B(w,b)]$

Average Coil Scheme 3: $CNND = \frac{d_A(w,a)+d_B(w,b)}{2}$

In all schemes, if $CNND > \tau$, then the query word is classified as coil.

The Min Coil Scheme is used most extensively in this investigation. If the query word is determined not to be coil because it is above τ, then classification as alpha or beta occurs. There are three variants for voting alpha vs. beta:

1-Nearest Neighbor Classification (1NN): If $d_A(w,a) < d_B(w,b)$, where a is the closest alpha word and b is the closest beta word, classify the query word as alpha. Otherwise, classify the query word as beta. In the event that both distances are equal, the word is classified as beta. This is the default voting scheme.

k-Nearest Neighbor Classification (kNN): Gather the k alpha or beta nearest neighbors to the query word, where k is an odd integer. Within these nearest neighbors, if the total number of alpha words is greater than the total number of beta words, then classify the query word as alpha. Otherwise, classify the query word as beta. Since k is odd, there is no chance that the number of alpha words will equal the number of beta words.

k-Nearest Neighbor Classification with Radius τ (kNNR): Gather the k alpha or beta nearest neighbors to the query word, where k is an odd integer, but discard nearest neighbors whose distance to the query word is greater than τ. Within the leftover nearest neighbors, if the total number of alpha words is greater than the total number of beta words, then classify the query word as alpha. Otherwise, classify the query word as beta. If there is an odd number of words discarded, then there is a chance that the number of leftover alpha nearest neighbors will equal the number of leftover beta nearest neighbors.

Both the kNN and kNNR voting variants are based on a majority-vote. Each word has a vote with a value of 1 in the set of k words, or k minus the number of discarded words. The value that a vote is worth can be weighted, and this is implemented in two ways:

- *Weight by distance*: A weight of $\frac{1}{d_X(w,v)}$ applied to all $v \in k$ *nearest neighbors*, where w is the query word and X is the class of v.

- *Weight by touchstone*: A weight of $\frac{1}{|\mathbb{X}|}$ applied to all $v \in k$ *nearest neighbors*, where w is the query word and $|\mathbb{X}|$ is the number of training words in the class of v. The set of training words constitutes the words used to train the distance function.

As mentioned above, in the kNNR voting scheme there is a chance that the numbers of alpha nearest neighbors and beta nearest neighbors in the set of k nearest neighbors are equal. The likelihood of this happening is attenuated by applying weights described above.

2.3 *Learning a Distance Function*

This section describes how a distance function is learned. The distance function between two words is the sum of the individual distances, or substitution scores, of amino acids with the same position in both words. A substitution score is dependent on its designated class, the identity of the amino acids, and the position in the word. k-Nearest Neighbor classification and linear programming are then used to generate a new distance function that replaces the default distance function. Iteration uses previously-generated distance functions to learn better distance functions.

2.3.1 *Touchstone and Training Sample Generation*

A database of known protein sequences and their associated secondary structures were obtained from the National Center for Biotechnology Information (NCBI) Reference Sequence Database[5]. The version of the database used in this investigation is uncertain, but is estimated to have been updated in 2012.

Each time that the code is run a single file of protein sequences is chosen to be the source of training and testing sequences. k-fold cross validation splits the sequences in to k equally-sized[1] protein subsets. In a given fold, k-1 of the protein subsets contain training proteins, which are used to generate fixed-length training words for the training set of words, or the touchstone. The remaining subset that contains testing proteins is used to generate fixed-length testing words for the testing set of words. Ultimately, k-fold cross validation generates k different combinations of the k subsets of training and testing proteins, and each combination is called a fold. Each fold designates a different testing protein subset out of the k protein subsets. The result is k distinct testing sets within the k folds, each paired with a touchstone made up of protein words that did not come from the testing protein subset. In a fold, it is possible that duplicate words exist between the touchstone and testing sets, since k-fold cross validation partitions whole protein sequences rather than protein words.

Words from each class in the touchstone are placed in separate files. The subset of alpha words within the touchstone will be called the alpha touchstone, symbolized by $\mathbb{A}$. Likewise, the subset of beta words will be called

[1] The subsets of protein sequences will only be equally-sized if the number of sequences is divisible by k. Otherwise, the subsets are approximately equally-sized.

the beta touchstone, symbolized by $\mathbb{B}$, and the subset of coil words will be called the coil touchstone, symbolized by $\mathbb{C}$. In a word, the central amino acid's secondary structure determines the class of the word, meaning that word lengths must always be odd. Words with invalid secondary structures, such as those in an alpha helix less than 5 amino acids long or a beta sheet less than 3 amino acids long, are removed from the touchstone. Additionally, duplicate words both from within and between each set were removed to ensure that each word in the touchstone for each class was unique internally to its own class and externally to the other classes. Thus, the overarching touchstone contains no duplicate words.

The touchstone for each class is typically quite large, which slows down computation. In order to attenuate running time of the code, a random sample of training words, called a training sample, is taken from each touchstone to be used as a supply of training query words in further steps. The training sample for each class is written to its own file. In summary, the alpha, beta, and coil touchstones comprise the overall touchstone, and are symbolized by $\mathbb{A}, \mathbb{B}$, and $\mathbb{C}$. The alpha, beta, and coil training samples contain random subsets of words from $\mathbb{A}, \mathbb{B}$, and $\mathbb{C}$, respectively, and will be symbolized by T_A, T_B, and T_C.

2.3.2 *Nearest Neighbor Querying*

The nearest neighbors for words in each training sample are found. For words from the training samples T_A and T_B, k nearest neighbors from the same class are found from either the alpha or beta touchstones. These nearest-neighbors are termed *target* words. Likewise, l nearest neighbors are taken from the opposite touchstone, which can be either alpha or beta but not coil. These nearest-neighbors are termed *imposter* words. Words from T_A and T_B and their corresponding targets and impostors are written to separate files for a total of four files:

- alpha training words and their alpha targets (a-a)
- alpha training words and their beta imposters (a-b)
- beta training words and their beta targets (b-b)
- beta training words and their alpha imposters (b-a)

For each coil word, l impostor nearest-neighbors are found from both the alpha and beta touchstones. Words in T_C and their corresponding impostors are also written to separate files:

- coil training words and their alpha imposters (c-a)
- coil training words and their beta imposters (c-b)

Note that although query words come from the training samples, the nearest neighbors are queried from the touchstone, and are not restricted to the training sample.

2.3.3 *The Dispersion Tree*

Protein words with known alpha or beta classifications are stored in separate data structures, called dispersion trees, which are queried for nearest neighbors. A dispersion tree is a tree-like data structure that takes $O(n^{1.5}logn)$ time to construct, where n is the number of words stored in the tree. For distance functions that satisfy the triangle inequality (Eq. 2), dispersion trees provide the fastest nearest-neighbor search time of any available data structure (Woerner, 2016)[6].

2.3.4 *Linear Program Formulation*

The linear program (LP) is subject to four types of inequalities: the triangle inequality, the identity inequality, the non- inequality, and various threshold inequalities. The triangle, identity, and non-negativity inequalities act on substitution score variables, while the threshold inequalities act on error variables and are constructed using the words in the training sample and their nearest neighbors. The objective of the linear program is to minimize the sum of the error variables subject to the constraints set by the inequalities. Upon completing the minimization, the values of the substitution scores constitute the improved distance function. The triangle, identity, and non-negativity inequalities are listed below.

- Triangle inequality:

$$S_{X,i}(a,b) + S_{X,i}(b,c) \geq S_{X,i}(a,c) \tag{2}$$

  for $X \in$ *classes*, $i \in$ *positions*, and $a, b, c \in$ *the set of amino acids*

- Identity inequality:

$$S_{X,i}(a,b) + \geq S_{X,i}(a,a) \tag{3}$$

  for $X \in$ *classes*, $i \in$ *positions*, and $a, b \in$ *the set of amino acids*

- Non-negativity inequality:

$$S_{X,i}(a,b) \geq 0 \tag{4}$$

  for $X \in$ *classes*, $i \in$ *positions*, and $a, b \in$ *the set of amino acids*

The threshold inequalities are constructed using the files of query words and their targets or imposters that were created in the nearest-neighbor querying step. Additionally, there exist two threshold variables τ_A and τ_B whose values are set by the linear program.

For a word $w \in T_A$:

$$d_A(w, v_A) \leq \tau_A \quad \text{for } k\ v_A \text{ targets } \in \mathbb{A} - w \tag{5}$$

$$d_B(w, v_B) \geq \tau_B + 1 \quad \text{for } l \; v_B \text{ imposters } \in \mathbb{B} \tag{6}$$

For a word $w \in T_B$:

$$d_A(w, v_A) \geq \tau_A + 1 \quad \text{for } k \; v_A \text{ imposters } \in \mathbb{A} \tag{7}$$

$$d_B(w, v_B) \leq \tau_B \quad \text{for } l \; v_B \text{ targets } \in \mathbb{B} - w \tag{8}$$

For a word $w \in T_C$:

$$d_A(w, v_A) \geq \tau_A + 1 \quad \text{for } l \; v_A \text{ imposters } \in \mathbb{A} \tag{9}$$

$$d_B(w, v_B) \geq \tau_B + 1 \quad \text{for } l \; v_B \text{ imposters } \in \mathbb{B} \tag{10}$$

Ideally, the threshold inequalities would be completely satisfied in the solution to the linear program. Realistically, it is highly unlikely that every target from the touchstone will be below its threshold and every imposter from the touchstone will be above its threshold. Therefore, the threshold inequalities must be adjusted to allow for violations. The amount by which they are violated is quantified in error variables. The threshold inequalities can be reworked as follows:

For a word $w \in T_A$:

for $k \; v_A$ targets $\in \mathbb{A} - w$:

$$e_{w,v_A} \geq d_A(w, v_A) - \tau_A \tag{11}$$

$$e_{w,v_A} \geq 0 \tag{12}$$

for $l \; v_B$ imposters $\in \mathbb{B}$

$$e_w \geq \tau_B + 1 - d_B(w, v_B) \tag{13}$$

$$e_{w,v_B} \geq 0 \tag{14}$$

For a word $w \in T_B$:

for $k \; v_B$ targets $\in \mathbb{B} - w$

$$e_{w,v_B} \geq d_B(w, v_B) - \tau_B \tag{15}$$

$$e_{w,v_B} \geq 0 \tag{16}$$

for l v_A imposters $\in \mathbb{A}$

$$e_w \geq \tau_A + 1 - d_A(w, v_A) \tag{17}$$

$$e_{w,v_A} \geq 0 \tag{18}$$

For a word $w \in T_C$:

for l v_A imposters $\in \mathbb{A}$

$$e_w^A \geq \tau_A + 1 - d_A(w, v_A) \tag{19}$$

$$e_w^A \geq 0 \tag{20}$$

for l v_B imposters $\in \mathbb{B}$

$$e_w^A \geq \tau_B + 1 - d_B(w, v_B) \tag{21}$$

$$e_w^A \geq 0 \tag{22}$$

The last constraint that is given to the linear program is that $\tau_A = \tau_B$. Finally, an objective function that minimizes error is set up from the error variables, and is split into alpha, beta, and coil components:

$$\frac{\beta_A}{|\mathbb{A}|} \sum_{w \in T_A} (\frac{\alpha}{k} \sum_{k \text{ targets } from \ \mathbb{A}} e_{w,v_A} + (1-\alpha)e_w) \quad +$$

$$\frac{\beta_B}{|\mathbb{B}|} \sum_{w \in T_B} (\frac{\alpha}{k} \sum_{k \text{ targets } from \ \mathbb{B}} e_{w,v_B} + (1-\alpha)e_w) \quad +$$

$$\frac{1-\beta_A-\beta_B}{|\mathbb{C}|} \sum_{w \in T_C} (e_w^A + e_w^B) \tag{23}$$

β_A, β_B and α are constants that were determined experimentally. β_A and β_B weight the alpha and beta errors versus the coil error. α weights the error on targets versus that on imposters. The overall objective function is to minimize the sum of the total error (Eq. 23).

In summary, the LP contains the inequalities listed above (Eq. 2-4 and 11-22), with the objective being to minimize the amount that the threshold inequalities are violated, which is quantified through the error variables.

2.3.5 *Linear Program Optimization and Iteration*

The above objective function and set of constraints are input into a linear program to be solved by the CPLEX Optimization Studio. CPLEX forms a matrix from the constraints listed in section 2.3.4. This matrix is large and sparse due to the large amount of inequalities and variables that make up the constraints. CPLEX accepts various parameters before optimizing, such as which optimizer to use to solve the linear program. There are various optimizers to select from, each suited to linear programming problems with certain characteristics. For our purposes, the barrier optimizer was deemed appropriate due to it being suited for large, sparse matrices. Additionally, barrier crossover was deactivated.

The output from CPLEX is a new set of substitution scores, and therefore a new function. The new distance function will replace the old distance function, and one of two things can occur next. The new distance function can be used to predict the secondary structure classifications of a set of testing words; or, the nearest-neighbor querying, LP formulation, and LP optimization steps can be repeated with the new distance function in order to generate an even better distance function. Nearest-neighbor querying, LP formulation, and LP optimization steps can occur for however many iterations as seen fit.

2.4 *Finding an Optimal Coil/Non-coil Threshold*

A value of $\tau = \tau_A = \tau_B$ is output from the linear program, but this initial value can be improved. An ideal value of τ would be one in which all words in the training sample below tau have true classifications as either alpha or beta, and all words in the training sample above tau have true classifications as coil. Of course, it is unlikely that training sample words will exhibit such binary behavior, regardless of which coil prediction scheme is used to classify the query words. Thus, the goal is to maximize the number of words above tau with true coil classifications and minimize the number of words below tau with true coil classifications. This can be achieved with a receiver operating characteristic, or ROC curve.

For each word $w \in T_X$ for $X \in classes$ and nearest neighbors $a \in \mathbb{A} - a$ and $b \in \mathbb{B} - b$, calculate the CNND of w, or in other words, either $min[d_A(w,a),\ d_B(w,b)]$, $max[d_A(w,a),\ d_B(w,b)]$, or $\frac{d_A(w,a)+d_B(w,b)}{2}$ depending on whether the minimum, maximum, or average coil schemes will be used for prediction, respectively (see section 2.2).

Next, the query words are ordered from lowest to highest CNND. For every pair of consecutive words that have different CNND values c_i and c_j, set a temporary value τ_{temp}:

$$\tau_{temp} = \frac{c_i + c_j}{2} \tag{24}$$

As with regular τ, words above τ_{temp} are predicted to be coil, whereas words below τ_{temp} are predicted to be non-coil. For every value of τ_{temp}, determine the following:

- Number of true positives, TP. This is the number of query words that lie above τ_{temp} and also have true coil classifications
- Number of true negatives, TN. This is the number of query words that lie below τ_{temp} and also have true non-coil classifications
- Number of false positives, FP. This is the number of query words that lie above τ_{temp}, but have true non-coil classifications
- Number of false negatives, FN. This is the number of query words that lie below τ_{temp}, but have true coil classifications

The true positive rate (TPR) and false positive rate (FPR) of a given τ_{temp} can be calculated:

$$TPR = \frac{TP}{TP + FN} \tag{25}$$

$$FPR = \frac{TN}{TN + FP} \tag{26}$$

Graphing TRP vs. the FPR for each value of τ_{temp} yields an ROC curve (Fig. 2). The upwardly-curved line indicates that the true positive rate at any given τ_{temp} is better than the false positive rate. The optimal tau will thereby be set to the value of τ_{temp} at which TPR + FPR = 1, which is the point on the curve that is closest to the point (0,1), at which the TPR is maximized while the FPR is minimized.

The area under the ROC curve is an indicator of the quality of the distance function. The more area under the curve, the closer the curve gets to the upper left corner of the graph, ultimately maximizing the TPR while keeping the FPR minimized at the point on the curve where TPR + FPR = 1. Given an optimal τ, a improved distance function should lead to improved classification of coil vs. non-coil. The ROC curve is therefore a useful debugging tool for learning the distance function, since each iteration of learning the distance function should yield to an ROC curve with increased area underneath it.

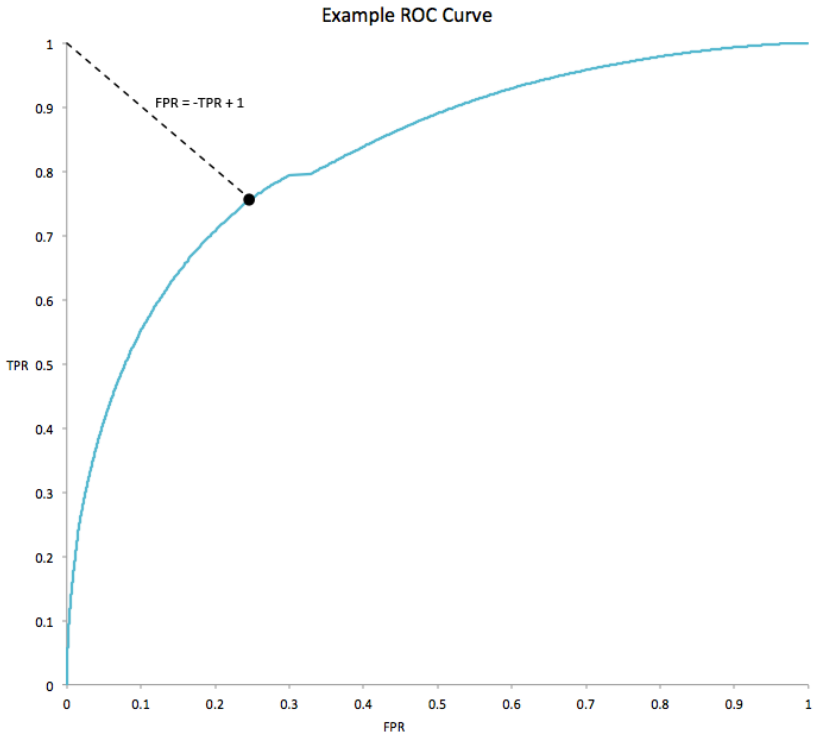

Figure 2: Sample ROC curve. Where the dashed line intersects the curve indicates the point on the curve where TPR + FPR = 1. This is the closest point on the curve to the upper-left corner of the graph.

3 EXPERIMENTAL RESULTS

3.1 *Input Parameters*

10-fold cross-validation was performed to break up a database of 5519 proteins into the touchstone and testing set. In each fold, 90% of the proteins were designated as training proteins, while 10% were designated as testing proteins. Both the touchstone and testing set were comprised of unique words of length 23, and words with invalid secondary structures were discarded, namely those in alpha helices or beta sheets with too few amino acids. Thus, although there could be no duplicate words within the touchstone and testing set individually, duplicate words could exist between both sets. In each fold, there were approximately 1,400,000 words in the touchstone. Approximately 460,000 of these words were known to be alpha, 280,000 were known to be beta, and 630, 000 were known to be coil. In each fold, there were approximately 270,000 words in the testing set. Approximately 100,000 of these words were known to be alpha, 60,000 were known to be beta, and 110,000 were known to be coil. The identities of the testing words were only used to evaluate our predictions.

150,000 words were randomly selected from the touchstone to be in the

training sample for each fold. The number of targets and imposters for a query word was set to 2 and 100, respectively. Lastly, 5 iterations were performed on each fold to improve the distance function.

3.2 *Determining Objective Function Constants β_A, β_B, and α*

In order to determine the optimal values for the objective function constants in the linear program, the distance function was learned nine different times, each for the same amount of iterations and with objective functions that had different combinations of $\beta_A = \beta_B$, and α (Fig. 3).

	$\beta_A = \beta_B$	α
Combination 1:	$\frac{1}{4}$	$\frac{1}{2}$
Combination 2:	$\frac{1}{4}$	$\frac{2}{3}$
Combination 3:	$\frac{1}{4}$	$\frac{4}{5}$
Combination 4:	$\frac{2}{5}$	$\frac{1}{2}$
Combination 5:	$\frac{2}{5}$	$\frac{2}{3}$
Combination 6:	$\frac{2}{5}$	$\frac{4}{5}$
Combination 7:	$\frac{1}{4}$	$\frac{1}{2}$
Combination 8:	$\frac{1}{4}$	$\frac{2}{3}$
Combination 9:	$\frac{1}{4}$	$\frac{4}{5}$

Figure 3: The different combinations of constants experimentally tried.

The ROC curves for each distance function were compared, and the objective function constants corresponding to the curve with the largest area underneath it were selected as the best constants to use out of the 9 combinations. The optimal constants were determined to be $\beta_A = \beta_B = \frac{1}{4}$, and $\alpha = \frac{1}{2}$ (Fig. 4). No tests were performed to vary β_A from β_B, but these tests should be performed in the future.

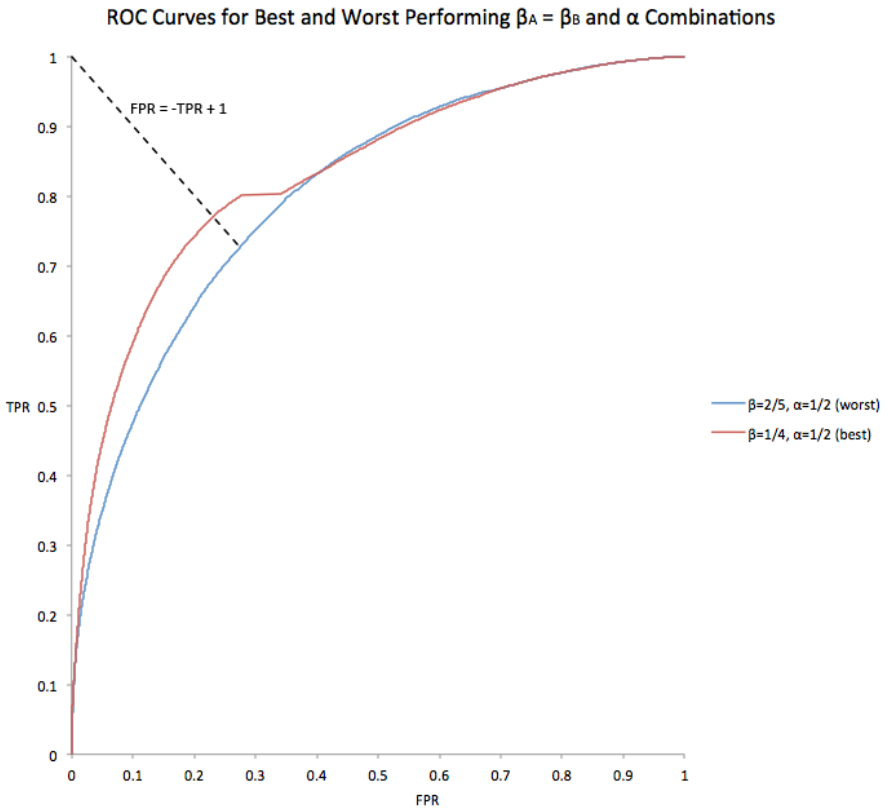

Figure 4: The area under the ROC curve corresponding to $\beta_A = \beta_B = 1/4$ and $\alpha = 1/2$ is the highest. The point on the curve that intersects with TPR + FPR = 1 is also the closest to the upper left corner of the graph. The area under the ROC curve corresponding to $\beta_A = \beta_B = 2/5$ and $\alpha = 1/2$ is the lowest out of all combinations of constants.

3.3 *ROC Curves for the Min Coil Scheme*

The classification of coil words becomes more accurate in each iteration of learning the distance function. Improvement slows considerably by the fifth iteration (Fig. 5).

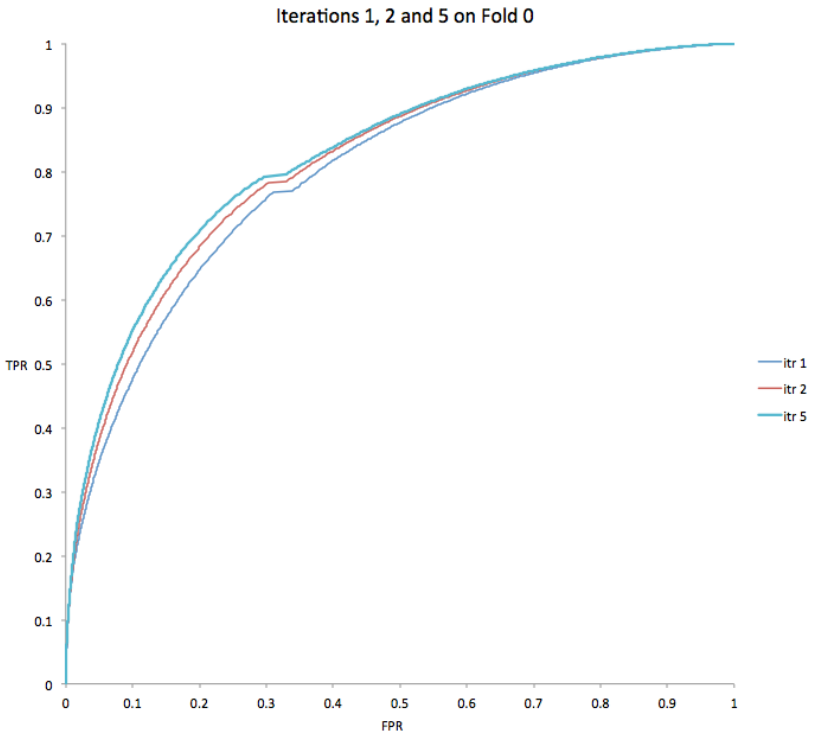

Figure 5: ROC curves for fold 0 of the tests run with the min coil scheme. The area under the ROC curves increases in each iteration, indicating improvement of coil classification in the distance function. Iterations 3 and 4 are omitted.

3.4 *Accuracies for Min Coil Scheme*

For the default coil/non-coil classification scheme, different alpha-beta voting variants were tested. One test using the 1-NN variant was performed. All different combinations of kNN and kNNR classification systems were performed by varying k from 3 to 5, and varying the weights applied to the voting. The best results were selected based on the highest class-wise and position-wise testing accuracies. The best results were obtained for 5-nearest neighbor with radius τ classification, weighted by distance (Fig. 6). Using this voting scheme, the training accuracies obtained were 67.3% for alpha, 62.6% for beta, 75.5% for coil, and 68.5% averaged. The testing accuracies obtained were 63.1% for alpha, 56.6% for beta, 71.6% for coil, 63.8% as the class-wise average, and 65.2% as the position-wise average.

Some accuracies obtained seem odd, however, such as those obtained for the 5-nearest neighbor classification scheme weighted by the touchstone, in which the alpha accuracies are significantly lower than normal while the beta accuracies are significantly higher than normal. A similar tendancy is exhibited in the 5-nearest neighbor with radius τ classification scheme weighted by the touchstone. This can likely be attributed to the low floating point precision

used for the weights that are applied to the votes. The value of $\frac{1}{|\mathbb{X}|}$, where $\mathbb{X}$ is the size of the alpha or beta touchstone, is very small. Even if the difference in size between both touchstones is small, the low floating point precision could exacerbate this difference. Future implementation of the weighting schemes should increase the floating point precision.

		training				testing				
variant	vote weight	α	β	coil	average	α	β	coil	class-wise ave	position-wise ave
1-NN	*none*	67.57%	63.06%	75.50%	68.71%	62.94%	56.45%	71.61%	63.67%	65.07%
3-NN	*none*	66.60%	61.87%	75.50%	67.99%	62.96%	56.37%	71.61%	63.64%	65.06%
3-NN	*touchstone*	66.60%	61.87%	75.50%	67.99%	62.96%	56.37%	71.61%	63.64%	65.06%
3-NN	*distance*	66.60%	61.87%	75.50%	67.99%	62.96%	56.37%	71.61%	63.64%	65.06%
3-NNR	*none*	65.97%	63.51%	75.50%	68.33%	61.75%	57.56%	71.61%	63.64%	64.88%
3-NNR	*touchstone*	65.97%	63.51%	75.50%	68.33%	61.75%	57.56%	71.61%	63.64%	64.88%
3-NNR	*distance*	67.32%	62.67%	75.50%	68.50%	63.07%	56.53%	71.61%	63.73%	65.14%
5-NN	*none*	66.08%	61.30%	75.50%	67.63%	62.83%	56.46%	71.61%	63.63%	65.03%
5-NN	*touchstone*	59.54%	66.23%	75.50%	68.09%	56.66%	61.09%	71.61%	63.12%	63.78%
5-NN	*distance*	66.08%	61.30%	75.50%	68.63%	62.83%	56.46%	71.61%	63.63%	65.03%
5-NNR	*none*	65.53%	63.62%	75.50%	68.22%	61.41%	57.84%	71.61%	63.62%	64.82%
5-NNR	*touchstone*	64.27%	64.45%	75.50%	68.07%	60.27%	58.56%	71.61%	63.48%	64.55%
5-NNR	*distance*	67.26%	62.59%	75.50%	68.45%	63.08%	56.57%	71.61%	**63.75**%	**65.15**%

Figure 6: Training and testing accuracies for the min coil scheme and all possible alpha-beta voting schemes.

3.5 *Preliminary Results for Max and Average Coil Schemes*

The max and average coil schemes were not the focus of this investigation, but preliminary results for these schemes were obtained for only one fold, and only for the 1NN alpha-beta voting scheme.

Using the max coil scheme and 1NN alpha-beta voting, the training accuracies obtained were 65.0% for alpha, 60.7% for beta, 74.7% for coil, and 66.8% averaged. The testing accuracies obtained were 59.5% for alpha, 53.2% for beta, 74.3% for coil, 62.3% as the class-wise average, and 64.2% as the position-wise average.

Using the average coil scheme and 1NN alpha-beta voting, the training accuracies obtained were 54.3% for alpha, 50.5% for beta, 66.3% for coil, and 57.0% averaged. The testing accuracies obtained were 51.2% for alpha, 46.6% for beta, 67.6% for coil, 55.2% as the class-wise average, and 56.9% as the position-wise average.

3.6 *Runtime*

On the University of Arizona High Performance Computing System, running 5 iterations to improve the distance function takes about 48 hours for each fold. Predicting the secondary structures of each testing set of approximately 270,000 words takes about 4 hours. Thus, predicting the secondary structure of each amino acid in a protein with 350 amino acids will take about 18 seconds. This does not include the X minutes needed to load the touchstone of 150,000 words into the partition tree.

4 DISCUSSION

In our version of single-sequence protein secondary structure prediction, a learned distance function determines the distance between a query word and words in either the alpha or beta touchstones. The distance function does not have a coil component, however, meaning that the distance between the query word and known coil words cannot be found. Instead, if a query word is sufficiently far from its alpha and beta nearest neighbors – that is, its critical nearest neighbor distance is above the coil/non-coil threshold τ – then the query word is classified as coil. The coil/non-coil threshold exists because words cannot be classified as coil based on nearest-neighbor classification. This is because the coil class is unstructured, containing all structures not alpha or beta. Thus, the number of possible coil words is large, and it may not be practical, or even possible, to have a touchstone large enough to contain them all.

An advantage of our approach is that once a distance function is learned, the secondary structures of protein words can be predicted quickly, and without using a protein database. Rather, a touchstone of words is stored in a

partition tree, and construction of the tree and querying words in the tree are both efficient operations. Additionally, the size of the training sample is an adjustable parameter that has only been briefly addressed, and smaller sizes can decrease the runtime of learning the distance function. Given a touchstone with a fixed cardinality, training sample cardinalities $|T_A|$ and $|T_B|$ can be made as low as necessary. Our first attempts of learning a distance function via our current approach had $|T_A| = |T_B| = 60,000$ query words, and the runtime for learning the distance function was considerably decreased due to the nearest-neighbor querying step taking less time and there being fewer inequalities being input into CPLEX. Lower training sample cardinalities yield lower prediction accuracies, however.

In future work, fixed word lengths other than 23 will be tested in order to determine the optimal word length. Word lengths of 21, 19, and 17 should be tested in order to see the effects of decreased word lengths on prediction accuracies. Although it is expected that lower word lengths will decrease the training accuracies, the distance function will not over-fit to the training data. As a result, the distance function may generalize better to the testing data.

Furthermore, global prediction of secondary structure can be implemented that takes into account an entire protein sequence when predicting the secondary structure of a word. Classification of a word would be partially based on the secondary structures of its adjacent words. Doing this will avoid predicting alpha helix or beta sheet strands that are too short. Dynamic programming might allow for optimal prediction that avoids these short runs. Another goal is to predict using confidences that query words are in certain classes, rather than our current all-or-nothing classification approach.

More work can be done fine-tuning the parameters that are set before learning the distance function. The values of k for the number of targets and l for the number of targets were constant at 2 and 100, respectively. Similarly, CPLEX was always run with the same parameters, which are detailed in section 2.3.5. However, CPLEX offers many more parameters that can be tuned. It is expected that adjusting additional parameters would affect the solving of the linear program.

Lastly, the non-redundant database of protein sequences compiled by NCBI is updated frequently. It is expected that the database of proteins used in our investigation is a few years old, and can be updated in future versions of our prediction code.

REFERENCES

[1] Aslanzadeh, Vahid, and Mostafa Ghaderia. "Homology Modeling and Functional Characterization of PR-1a Protein of Hordeum Vulgare Subsp. Vulgare." Bioinformation 8.17 (2012): 807-11. Web.

[2] Benjamin Yee, Personal communication, 2014

[3] IBM Corp. Released 2015. IBM ILOG CPLEX Optimization Studio, Version

12.6.2 Armonk, NY: IBM Corp.

[4] Jones, David T. "Protein Secondary Structure Prediction Based on Position-specific Scoring Matrices." Journal of Molecular Biology 292.2 (1999): 195-202. Web.

[5] The NCBI handbook [Internet]. Bethesda (MD): National Library of Medicine (US), National Center for Biotechnology Information; 2002 Oct. Chapter 18, The Reference Sequence (RefSeq) Project. Available from http://www.ncbi.nlm.nih.gov/books/NBK21091/

[6] Woerner, August, and John Kececioglu. "Faster metric-space nearest-neighbor search using dispersion trees." In preparation, 2016.