Protein Fold Recognition Using the Gradient Boost Algorithm

Feng Jiao*
School of Computer Science, University of Waterloo, Canada
fjiao@cs.uwaterloo.ca

Jinbo Xu†
Toyota Technological Institute at Chicago, USA
j3xu@tti-c.org

Libo Yu
Bioinformatics Solutions Inc., Waterloo, Canada
libo@bioinformaticssolutions.com

Dale Schuurmans
Department of Computing Science, University of Alberta, Canada
dale@cs.ualberta.ca

Protein structure prediction is one of the most important and difficult problems in computational molecular biology. Protein threading represents one of the most promising techniques for this problem. One of the critical steps in protein threading, called fold recognition, is to choose the best-fit template for the query protein with the structure to be predicted. The standard method for template selection is to rank candidates according to the z-score of the sequence-template alignment. However, the z-score calculation is time-consuming, which greatly hinders structure prediction at a genome scale. In this paper, we present a machine learning approach that treats the fold recognition problem as a regression task and uses a least-squares boosting algorithm (LS_Boost) to solve it efficiently. We test our method on Lindahl's benchmark and compare it with other methods. According to our experimental results we can draw the conclusions that: (1) Machine learning techniques offer an effective way to solve the fold recognition problem. (2) Formulating protein fold recognition as a regression rather than a classification problem leads to a more effective outcome. (3) Importantly, the LS_Boost algorithm does not require the calculation of the z-score as an input, and therefore can obtain significant computational savings over standard approaches. (4) The LS_Boost algorithm obtains superior accuracy, with less computation for both training and testing, than alternative machine learning approaches such as SVMs and neural networks, which also need not calculate the z-score. Finally, by using the LS_Boost algorithm, one can identify important features in the fold recognition protocol, something that cannot be done using a straightforward SVM approach.

1. INTRODUCTION

In the post-genomic era, understanding protein function has become a key step toward modelling complete biological systems. It has been established that the functions of a protein are directly linked to its three-dimensional structure. Unfortunately, current "wet-lab" methods used to determine the three-dimensional structure of a protein are costly, time-consuming and sometimes unfeasible. The ability to predict a protein's structure directly from its sequence is urgently needed in the post-genomic era, where protein sequences are becoming available at a far greater rate than the corresponding structure information.

Protein structure prediction is one of the most important and difficult problems in computational molecular biology. In recent years, *protein threading* has turned out to be one of the most successful approaches to this problem [7, 14, 15]. Protein threading predicts protein structures by using statistical knowledge of the relationship between protein sequences and structures. The prediction is made by aligning each amino acid in the target sequence to a position in a template structure and evaluating how well the target fits the template. After aligning the se-

*Work performed at the Alberta Ingenuity Centre for Machine Learning, University of Alberta.
†Contact author.

quence to each template in the structural template database, the next step then is to separate the correct templates from incorrect templates for the target sequence—a step we refer to as *template selection* or *fold recognition*. After the best-fit template is chosen, the structural model of the sequence is built based on the alignment between the sequence and the chosen template.

The traditional fold recognition technique is based on calculating the z-score, which statistically tests the possibility of the target sequence folding into a structure very similar to the template [3]. In this technique, the z-score is calculated for each sequence-template alignment by first determining the distribution of alignment scores among random re-shufflings of the sequence, and then comparing the alignment score of the correct sequence (in standard deviation units) to the average alignment score over random sequences. Note that the z-score calculation requires the alignment score distribution to be determined by randomly shuffling the sequence many times (approx. 100 times), meaning that the shuffled sequence has to be threaded to the template repeatedly. Thus, the entire process of calculating the z-score is very time-consuming. In this paper, instead of using the traditional z-score technique, we propose to solve the fold recognition problem by treating it as a machine learning problem.

Several research groups have already proposed machine learning methods, such as neural networks [9, 23] and support vector machines (SVMs) [20, 22] for fold recognition. In this general framework, for each sequence-template alignment, one generates a set of features to describe the instance, treats the extracted features as input data, and the alignment accuracy or similarity level as a response variable. Thus, the fold recognition problem can be expressed as a standard prediction problem that can be solved by supervised machine learning techniques for regression or classification. In this paper we investigate a new approach that proves to be simpler to implement, more accurate and more computationally efficient. In particular, we combine the gradient boosting algorithm of Friedman [5] with a least-squares loss criterion to obtain a least-squares boosting algorithm, LS_Boost. We use LS_Boost to estimate the alignment accuracy of each sequence-template alignment and employ this as part of our fold recognition technique.

To evaluate our approach, we experimentally test it on Lindahl's benchmark [12] and compare the resulting performance with other fold recognition methods, such as the z-score method, SVM regression, SVM classification, neural networks and Bayes classification. Our experimental results demonstrate that the LS_Boost method outperforms the other techniques in terms of both prediction accuracy and computational efficiency. It is also a much easier algorithm to implement.

The remainder of the paper is organized as follows. We first briefly introduce the idea of using protein threading for protein structure prediction. We show how to generate features from each sequence-template alignment and convert protein threading into a standard prediction problem (making it amenable to supervised machine learning techniques). We discuss how to design the least-squares boosting algorithm by combining gradient boosting with a least-squares loss criterion, and then describe how to use our algorithm to solve the fold recognition problem. Finally, we will describe our experimental set-up and compare LS_Boost with other methods, leading to the conclusions we present in the end.

2. Protein Threading and Fold Recognition

2.1. The threading method for protein structure prediction

The idea of protein threading originated from the observation that the number of different structural folds in nature may be quite small, perhaps two orders of magnitude fewer than the number of known protein sequences [11]. Thus, the structure prediction problem can be potentially reduced to a problem of recognition: choosing a known structure into which the target sequence will fold. Or, put another way, protein threading is in fact a database search technique, where given a query sequence of unknown structure, one searches a structure (template) database and finds the best-fit structure for the given sequence. Thus, protein threading typically consists of the following four steps:

(1) Build a template database of representative three-dimensional protein structures, which usu-

ally involves removing highly redundant structures.

(2) Design a scoring function to measure the fitness between the target sequence and the template based on the knowledge of the known relationship between the structures and the sequences. Usually, the minimum value of the scoring function corresponds to the optimal sequence-template alignment.
(3) Find the best alignment between the target sequence and the template by minimizing the scoring function.
(4) Choose the best-fit template for the sequence according to a criterion, based on all the sequence-template alignments.

In this paper, we will only focus on the final step. That is, we only discuss how to choose the best template for the sequence, which is called fold recognition. We use our existing protein threading server RAPTOR [21, 22] to generate all the sequence-structure alignments. For the fold recognition problem, there are two different approaches: the z-score method [3] and the machine learning method [9, 23].

2.2. The z-score method for fold recognition

The z-score is defined to be the "distance" (in standard deviation units) between the optimal alignment score and the mean alignment score obtained by randomly shuffling the target sequence. An accurate z-score can cancel out the sequence composition bias and offset the mismatch between the sequence size and the template length. Bryant et al. [3] proposed the following procedures to calculate z-score:

(1) Shuffle the aligned sequence residues randomly.
(2) Find the optimal alignment between the shuffled sequence and the template.
(3) Repeat the above two steps N times, where N is on the order of one hundred. Then calculate the distribution of these N alignment scores.

After the N alignment scores are obtained, we calculate the deviation of the optimal alignment score from the distribution of these N alignment scores.

We can see from above that in order to calculate the z-score for each sequence-template alignment, we need to shuffle and rethread the target sequence many times, which takes a significant amount of time and essentially prevents this technique from being applied to genome-scale structure prediction.

2.3. Machine learning methods for fold recognition

Another approach to the fold recognition problem is to use machine learning methods, such as neural networks, as in the GenTHREADER [9] and PROSPECT-I systems [23], or SVMs, as in the RAPTOR system [22]. Current machine learning methods generally treat the fold recognition problem as a classification problem. However, there is a limitation to the classification approach that arises when one realizes that there are three levels of similarity that one can draw between two proteins: fold level similarity, superfamily level similarity and family level similarity. Currently, classification-based methods treat the three different similarity levels as a single level, and thus are unable to effectively differentiate one similarity level from another while maintaining a hierarchical relationship between the three levels. Even a multi-class classifier cannot deal with this limitation very well since the three levels are in a hierarchical relationship.

Instead, we use a regression approach, which simply uses the alignment accuracy as the response value. That is, we reformulate the fold recognition problem as predicting the alignment accuracy of a threading pair, which then is used to differentiate the similarity level between proteins. In our approach, we use SARF [2] to generate the alignment accuracy between the target protein and the template protein. The alignment accuracy of threading pair is defined to be the number of correctly aligned positions, based on the correct alignment generated by SARF. A position is correctly aligned only if its alignment position is no more than four position shifts away from its correct alignment. On average, the higher the similarity level between two proteins, the higher the value of the alignment accuracy will be. Thus alignment accuracy can help to effectively differentiate the three similarity levels. Below we will show in our experiments that the regression approach obtains much better results than the standard classification approach.

3. Feature Extraction

One of the key steps in the machine learning approach is to choose a set of proper features to be used as inputs for predicting the similarity between two proteins. After optimally threading a given sequence to each template in the database, we generate the following features from each threading pair.

(1) Sequence size, which is the number of residues in the sequence.
(2) Template size, which is the number of residues in the template.
(3) Alignment length, which is the number of aligned residues. Usually, two proteins from the same fold class should share a large portion of similar sub-structure. If the alignment length is considerably smaller than the sequence size or the template size, then it indicates that this threading pair is unlikely to be in the same SCOP class.
(4) Sequence identity. Although a low sequence identity does not imply that two proteins are not similar, a high sequence identity can indicate that two proteins should be considered as similar.
(5) Number of contacts with both ends being aligned to the sequence. There is a contact between two residues if their spatial distance is within a given cutoff. Usually, a longer protein should have more contacts.
(6) Number of contacts with only one end being aligned to the sequence. If this number is big, then it might indicate that the sequence is aligned to an incomplete domain of the template, which is not good since the sequence should fold into a complete structure.
(7) Total alignment score.
(8) Mutation score, which measures the sequence similarity between the target protein and the template protein.
(9) Environment fitness score. This feature measures how well to put a residue into a specific environment.
(10) Alignment gap penalty. When aligning a sequence and a template, some gaps are allowed. However, if there are too many gaps, it might indicate that the quality of the alignment is bad, and therefore the two sequences may not be in the same similarity level.
(11) Secondary structure compatibility score, which measures the secondary structure difference between the template and the sequence in all positions.
(12) Pairwise potential score, which characterizes the capability of a residue to make a contact with another residue.
(13) The z-score of the total alignment score and the z-score of a single score item such as mutation score, environment fitness score, secondary structure score and pairwise potential score.

Notice that here we still take into consideration the traditional z-score for the sake of performance comparison. But later we will show that we can obtain nearly the same performance without using the z-score, which means it is unnecessary to calculate the z-score as one of the features.

We calculate the alignment accuracy between the target protein and the template protein using a structure comparison program SARF. We use the alignment accuracy as the response variable. Given the training set with input feature vectors and the response variable, we need to find a prediction function that maps the features to the response variable. By using this function, we can estimate the alignment accuracy for each sequence-template alignment. Then, all the sequence-template alignments can be ranked based on the predicted alignment accuracy and the first-ranked one is chosen as the best alignment for the sequence. Thus we have converted the protein structure problem to a function estimation problem. In the next section, we will show how to design our LS_Boost algorithm by combining the gradient boosting algorithm of Friedman [5] with a least-squares loss criterion.

4. Least-Squares Boosting Algorithm For Fold Recognition

The problem can be formulated as follows. Let x denote the feature vector and y the alignment accuracy. Given an input variable x, a response variable y and some samples $\{y_i, x_i\}_{i=1}^N$, we want to find a function $F^*(x)$ that can predict y from x such that over the joint distribution of $\{y, x\}$ values, the expected value of a specific loss function $L(y, F(x))$ is minimized [5].

The loss function is used to measure the deviation between the real y value and the predicted y value.

$$\begin{aligned} F^*(x) &= \arg\min_{F(x)} E_{y,x} L(y, F(x)) \\ &= \arg\min_{F(x)} E_x[E_y L(y, F(x))|x] \end{aligned} \qquad (1)$$

Normally $F(x)$ is a member of a parameterized class of functions $F(x; P)$, where P is a set of parameters. We use the form of the "additive" expansions to design the function as follows:

$$F(x; P) = \sum_{m=0}^{M} \beta_m h(x; \alpha_m) \qquad (2)$$

where $P = \{\beta_m, \alpha_m\}_{m=0}^{M}$. The functions $h(x; \alpha)$ are usually simple functions of x with parameters $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_M\}$. When we wish to estimate $F(x)$ non-parametrically the task becomes more difficult. In general, we can choose a parameterized model $F(x; P)$ and change the function optimization problem to parameter optimization. That is, we fix the form of the function and optimize the parameters instead. A typical parameter optimization method is a "greedy-stagewise" approach. That is, we optimize $\{\beta_m, \alpha_m\}$ after all of the $\{\beta_i, \alpha_i\}(i = 0, 1, \ldots, m-1)$ are optimized. This process can be represented by the following two recursive equations.

$$(\beta_m, \alpha_m) = \arg\min_{\beta,\alpha} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i; \alpha)) \qquad (3)$$

$$F_m = F_{m-1}(x) + \beta_m h(x; \alpha_m) \qquad (4)$$

Friedman proposed a steepest-descent method to solve the optimization problem described in Equation 2 [5]. This algorithm is called the Gradient Boosting algorithm and its entire procedure is given in Figure 1.

```
Algorithm 1: Gradient Boost
• Initialize $F_0(x) = \arg\min_\rho \sum_{i-1}^{N} L(y_i, \rho)$
• For $m = 1$ to $M$ do:
    • Step 1. Compute the negative gradient
        $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F_{x_i}}\right]$
    • Step 2. Fit a model
        $\alpha_m = \arg\min_{\alpha,\beta} \sum_{i=1}^{N} [\tilde{y} - \beta h(x_i; \alpha_m)]^2$
    • Step 3. Choose a gradient descent step size as
        $\rho_m = \arg\min_{\rho} \sum_{i-1}^{N} L(y_i, Fm-1(x_i)+\rho h(x_i; \alpha))$
    • Step 4. Update the estimation of $F(x)$
        $F_m(x) = F_{m-1}(x) + \rho_m h(x; \alpha_m)$
• end for
• Output the final regression function $F_m(x)$
```

Fig. 1. Gradient boosting algorithm

```
Algorithm 2: LS_Boost
• Initialize $F_0 = \bar{y} = \frac{1}{N} \sum_i y_i$
• For $m = 1$ to $M$ do:
    • $\tilde{y}_i = y_i - F_{m-1}(x_i, i = 1, \ldots, N)$
    • $(\rho_m, \alpha_m) = \arg\min_{\rho,\alpha} \sum_{i=1}^{N} [\tilde{y}_i - \rho h(x_i; \alpha_m)]^2$
    • $F_m(x) = F_{m-1}(x) + \rho_m h(x; \alpha_m)$
• end for
• Output the final regression function $F_m(x)$
```

Fig. 2. LS_Boost algorithm

By employing the least square loss function $(L(y, F)) = (y - F)^2/2$ we have a least-squares boosting algorithm shown in Figure 2. For this procedure,

ρ is calculated as follows:

$$(\rho, \alpha_m) = \arg\min_{\rho,\alpha} \sum_{i=1}^{N} [\tilde{y}_i - \rho h(x_i; \alpha_m)]^2$$

and therefore $$\rho = N \times \tilde{y}_i / \sum_{i=1}^{N} h(x_i; \alpha_m) \qquad (5)$$

The simple function $h(x, \alpha)$ can have any form that can be conveniently optimized over α. In terms of boosting, optimizing over α to fit the training data is called *weak learning*. In this paper, for considerations of speed, we choose some function for which it is easy to obtain α. The simplest function to use here is the linear regression function:

$$y = ax + b \qquad (6)$$

where x is the input feature and y is the alignment accuracy. The parameters of the linear regression function can be solved easily by the following equation:

$$a = \frac{l_{xy}}{l_{xx}}, b = y - ax$$

where $$l_{xx} = n \times \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2$$

$$l_{xy} = n \times \sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)$$

There are many other simple functions one can use, such as an exponential function $y = a + e^b x$, logarithmic function $y = a + b ln x$, quadratic function $y = ax^2 + bx + c$, or hyperbolic function $y = a + b/x$, etc.

In our application, for each round, we choose one feature and obtain the simple function $h(x, \alpha)$ with the minimum least-squares error. The underlying reasons for choosing a single feature at each round are: i) we would like to see the role of each feature in fold recognition; and ii) we notice that alignment accuracy is proportional to some features. For example, the higher the alignment accuracy, the lower the mutation score, fitness score and pairwise score. Figure 3 shows the relation between alignment accuracy and mutation score.

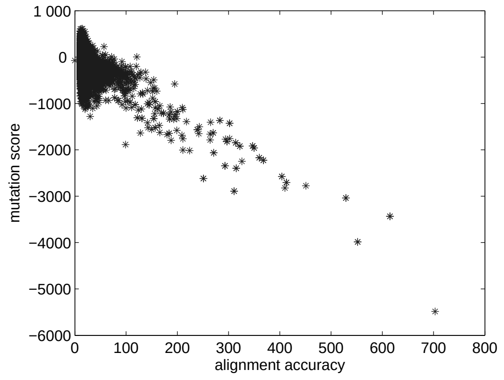

Fig. 3. The relation between alignment accuracy and mutation score.

In the end, we combine these simple functions to form the final regression function. As such, Algorithm 2 translates to the following procedures.

(1) Calculate the difference between the real alignment accuracy and the predicted alignment accuracy. We call this difference the alignment accuracy residual. Assume the initial predicted alignment accuracy is the average alignment accuracy of the training data.
(2) Choose a single feature which correlates best with the alignment accuracy residual. The parameter ρ is calculated by using Equation 5. Then the alignment accuracy residual is predicted by using this chosen feature and the parameter.
(3) Update the predicted alignment accuracy by adding the predicted alignment accuracy residual. Repeat the above two steps until the predicted alignment accuracy does not change significantly.

5. Experimental Results

When one protein structure is to be predicted, we thread its sequence to each template in the database and obtain the predicted alignment accuracy using the LS_Boost algorithm. We choose the template with the highest alignment accuracy as the basis to build the structure of the target sequence.

We can describe the relationship between two proteins at three different levels: the family level, superfamily level and the fold level. If two proteins are similar at the family level, then these two proteins have evolved from a common ancestor and usually share more than 30% sequence identity. If two proteins are similar only at the fold level, then their structures are similar even though their sequences are not similar. The superfamily-level similarity is something in between family level and fold level. If the target sequence has a template that is in the same family as the sequence, then it is easier to predict the structure of the sequence. If two proteins are similar only at fold level, it means they share less sequence similarity and it is harder to predict their relationship.

We use the SCOP database [16] to judge the similarity between two proteins and evaluate our predicted results at different levels. If the predicted template is similar to the target sequence at the family level according to the SCOP database, we treat it as correct prediction at the family level. If the predicted template is similar at the superfamily level but not at the family level, then we assess this prediction as being correct at the superfamily level. Similarly, if the predicted template is similar at the fold level but not at the other two levels, we assess the prediction as correct at the fold level. When we say a prediction is correct according to the top K criterion, we mean that there are no more than $K-1$ incorrect predictions ranked before this prediction. The fold-level relationship is the hardest to predict because two proteins share very little sequence similarity in this case.

To train the parameters in our algorithm, we randomly choose 300 templates from the FSSP list [1] and 200 sequences from Holm's test set [6]. By threading each sequence to all the templates, we obtain a set of 60,000 training examples.

To test the algorithm, we use Lindahl 's benchmark, which contains 976 proteins, each pair of which shares at most 40% sequence identity. By threading each one against all the others, we obtain a set of 976×975 threading pairs. Since the training set is chosen randomly from a set of non-redundant proteins, the overlap between the training set and Lindahl's benchmark is fairly small, which is no more than 0.4 percent of the whole test set. To ensure the complete separation of training and testing sets, these overlap pairs are removed from the test data. We calculate the recognition rate of each method at the three similarity levels.

5.1. Sensitivity

Figure 4 shows the sensitivity of our algorithm at each round. We can see that the LS_Boost algorithm nearly converges within 100 rounds, although we train the algorithm further to obtain higher performance.

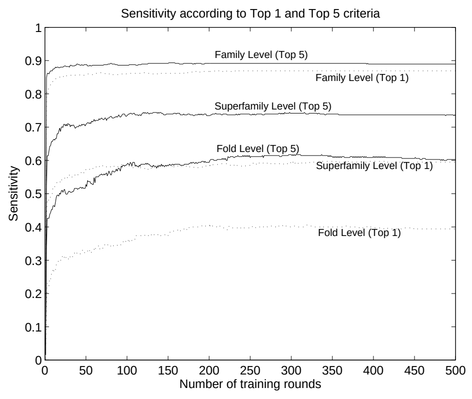

Fig. 4. Sensitivity curves during the training process.

Table 5.1 lists the results of our algorithm against several other algorithms. PROSPECT II uses the z-score method, and its results are taken from Kim et al.'s paper [10]. We can see that the LS_Boost algorithm is better than PROSPECT II at all three levels. The results for the other methods are taken from Shi et al's paper [18]. Here we can see that our method apparently outperforms the other methods. However, since we use different sequence-structure alignment methods, this disparity may be partially due to different threading techniques. Nevertheless, we can see that the machine learning approaches normally perform much better than the other methods.

Table 5.1 shows the results of our algorithm against several other popular machine learning methods. Here we will not describe the detail of each method. In this experiment, we use RAPTOR to generate all the sequence-template alignments. For each different method, we tune the parameters on the training set and test the model on the test set. In total we test the following six other machine learning

Table 1. Sensitivity of the LS_Boost method compared with other structure prediction servers.

	Family Level		Superfamily Level		Fold Level	
	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
RAPTOR (LS_Boost)	86.5%	89.2%	60.2%	74.4%	38.8%	61.7%
PROSPECT II	84.1 %	88.2%	52.6%	64.8%	27.7%	50.3%
FUGUE	82.3%	85.8%	41.9%	53.2%	12.5%	26.8%
PSI_BLAST	71.2%	72.3%	27.4%	27.9%	4.0%	4.7%
HMMER_PSIBLAST	67.7%	73.5%	20.7%	31.3%	4.4%	14.6%
SAMT98-PSIBLAST	70.1%	75.4%	28.3%	38.9%	3.4%	18.7%
BLASTLINK	74.6%	78.9%	29.3%	40.6%	6.9%	16.5%
SSEARCH	68.6%	75.7%	20.7%	32.5%	5.6%	15.6%
THREADER	49.2%	58.9%	10.8%	24.7%	14.6%	37.7%

Table 2. Performance comparison of seven machine learning methods. The sequence-template alignments are generated by RAPTOR.

	Family Level		Superfamily Level		Fold Level	
	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
LS_Boost	86.5%	89.2%	60.2%	74.4%	38.8%	61.7%
SVM (regression)	85.0%	89.1%	55.4%	71.8%	38.6%	60.6%
SVM (classification)	82.6%	83.6%	45.7%	58.8%	30.4%	52.6%
Ada_Boost	82.8%	84.1%	50.7%	61.1%	32.2%	53.3%
Neural Networks	81.1%	83.2%	47.4%	58.3%	30.1%	54.8%
Bayes classifier	69.9%	72.5%	29.2%	42.6%	13.6%	40.0%
Naïve Bayes Classifier	68.0%	70.8%	31.0%	41.7%	15.1%	37.4%

methods.

(1) SVM regression. Support vector machines are based on the concept of structural risk minimization from statistical learning theory [19]. The fold recognition problem is treated as a regression problem, therefore we consider SVMs used for regression. Here we use the svm_light software package [8] and an RBF kernel to obtain the best performance. As shown in Table 5.1, LS_Boost performs slightly better than SVM regression.
(2) SVM classification. The fold recognition problem is treated as a classification problem, and we consider an SVM for classification. The software and kernel we consider is the same as for SVM regression. In this case, one can see that SVM classification performs worse than SVM regression, especially at the superfamily level and the fold level.
(3) AdaBoost. Boosting is a procedure that combine the outputs of many "weak" classifiers to produce a powerful "committee". We use the standard AdaBoost algorithm [4] for classification, which is similar to LS_Boost except that it performs classification rather than regression and uses the exponential instead of least-squares loss function. The AdaBoost algorithm achieves a comparable result to SVM classification but is worse than both of the regression approaches, LS_Boost and SVM regression.
(4) Neural networks. Neural networks are one of the most popular methods used in machine learning [17]. Here we use a multi-layer perceptron for classification, based on the Matlab neural network toolbox. The performance of the neural network is similar to SVM classification and Adaboost.
(5) Bayesian classifier. A Bayesian classifier is a probability based classifier which assigns a sample to a class based on the probability that it belongs to the class [13].
(6) Naïve Bayesian classifier. The Naïve Bayesian classifier is similar to the Bayesian classifier except that it assumes that the features of each class are independent, which greatly decreases computation [13]. We can see both Bayesian classifier and Naïve Bayesian classifier obtain poor performance.

Our experimental results show clearly that: (1) The regression based approaches demonstrate better performance than the classification based approaches. (2) LS_Boost performs slightly better than

SVM regression and significantly better than the other methods. (3) The computational efficiency of LS_Boost is much better than SVM regression, SVM classification and the neural network.

One of the advantages of our boosting approach over SVM regression is its ability to identify important features, since at each round LS_Boost only chooses a single feature to approximate the alignment accuracy residual. The following are the top five features chosen by our algorithm. The corresponding simple functions associated with each feature are all linear regression functions $y = ax + b$, showing that there is a strong linear relation between the features and the alignment accuracy. For example, from the figure 3, we can see that the linear regression function is the best fit.

(1) Sequence identity;
(2) Total alignment score;
(3) Fitness score;
(4) Mutation score;
(5) Pairwise potential score.

It seems surprising that the widely used z-score is not chosen as one of the most important features. This indicates to us that the z-score may not be the most important feature and redundant. To confirm our hypothesis, we re-trained our model using all the features except all the z-scores. That is, we conducted the same training and test procedures as before, but with the reduced feature set. The results given in Table 3 show that for LS_Boost there is almost no difference between using the z-score as an additional feature or without using it. Thus, we conclude that by using the LS_Boost approach it is unnecessary to calculate z-score to obtain the best performance. This means that we can greatly improve the computational efficiency of protein threading without sacrificing accuracy, by completely avoiding the calculation of the expensive z-score.

To quantify the margin of superiority of LS_Boost over the other machine-learning methods, we use bootstrap method to get the error analysis. After training the model, we randomly sample 600 sequences from Lindahl's benchmark and calculate the sensitivity using the same method as before. We repeat the sampling for 1000 times and get the mean and standard deviation of the sensitivity of each method as listed in 5.1. We can see that LS_Boost method is slightly better than SVM regression and much better than other methods.

5.2. Specificity

We further examine the specificity of the LS_Boost method with Lindahl's benchmark. All threading pairs are ranked by their confidence score (i.e., the predicted alignment accuracy or the classification score if an SVM classifier is used) and the sensitivity-specificity curves are drawn in Figure 5, 6 and 7. Figure 6 demonstrates that at the superfamily level, the LS_boost method is consistently better than SVM regression and classification within the whole spectrum of sensitivity. At both the family level and fold level, LS_Boost is a little better when the specificity is high while worse when the specificity is low. At the family level, LS_Boost achieves a sensitivity of 55.0% and 64.0% at 99% and 50% specificities, respectively, whereas SVM regression achieves a sensitivity of 44.2% and 71.3%, and SVM classification achieves a sensitivity of 27.0% and 70.9% respectively. At the superfamily level, LS_Boost has a sensitivity of 8.2% and 20.8% at 99% and 50% specificities, respectively. In contrast, SVM regression has a sensitivity of 3.6% and 17.8%, and SVM classification has a sensitivity of 2.0% and 16.1% respectively. Figure 7 shows that at the fold level, there is no big difference between LS_Boost method, SVM regression and SVM classification method.

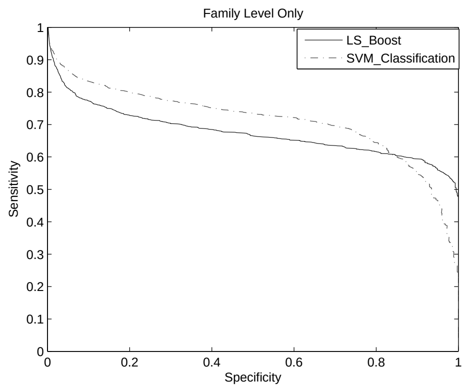

Fig. 5. Family-level specificity-sensitivity curves on Lindahl's benchmark set. The three methods LS_Boost, SVM regression and SVM classification are compared.

Table 3. Comparison of fold recognition performance with zscore and without zscore.

	Family Level		Superfamily Level		Fold Level	
	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
LS_Boost with z-score	86.5%	89.2%	60.2%	74.4%	38.8%	61.7%
LS_Boost without z-score	85.8%	89.2%	60.2%	73.9%	38.3%	62.9%

Table 4. Error Analysis of seven machine learning methods. The sequence-template alignments are generated by RAPTOR.

	Family Level				Superfamily Level				Fold Level			
	Top 1		Top 5		Top 1		Top 5		Top 1		Top 5	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
LS_Boost	86.57%	0.0290	89.15%	0.0305	60.17%	0.0294	74.29%	0.0342	38.86%	0.0273	61.75%	0.0362
SVM (regression)	85.15%	0.0309	89.15%	0.0307	55.57%	0.0290	71.97%	0.0329	38.68%	0.0269	60.70%	0.0349
SVM (classification)	82.49%	0.0276	83.76%	0.0298	45.75%	0.0264	58.86%	0.0304	30.45%	0.0244	52.80%	0.0321
Ada_Boost	82.94%	0.0296	84.22%	0.0291	50.74%	0.0279	61.26%	0.0308	32.18%	0.0254	53.40%	0.0336
Neural Networks	81.75%	0.0290	83.47%	0.0298	47.52%	0.0271	58.40%	0.0313	30.24%	0.0244	54.99%	0.0326
Bayes classifier	69.97%	0.0271	72.55%	0.0270	29.13%	0.0213	42.60%	0.0262	13.68%	0.0155	40.06%	0.0282
Naïve Bayes Classifier	68.77%	0.0261	70.97%	0.0277	31.05%	0.0216	41.87%	0.0248	15.10%	0.0166	37.34%	0.0270

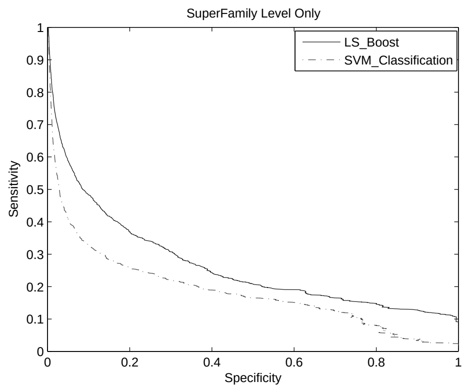

Fig. 6. Superfamily-level specificity-sensitivity curves on Lindahl's benchmark set. The three methods LS_Boost, SVM regression and SVM classification are compared.

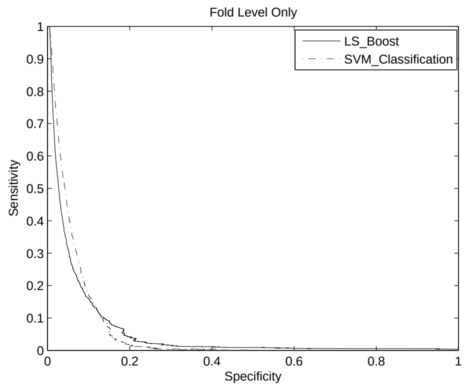

Fig. 7. Fold-level specificity-sensitivity curves on Lindahl's benchmark set. The three methods LS_Boost, SVM regression and SVM classification are compared.

5.3. Computational Efficiency

Overall, the LS_Boost procedure achieves superior computational efficiency during both training and testing. By running our program on a 2.53 GHz Pentium IV processor, after extracting the features, the training time is less than thirty seconds and the total test time is approximately two seconds. Thus we can see that our technique is very fast compared to other approaches, in particular the machine learning approaches such as neural networks and SVMs which require much more time to train. Table 5.3 lists the running time of several different fold recognition methods. From this table, we can see that the boosting approach is more efficient than the SVM regression method, which is desirable for genome-scale structure prediction. The running time shown in this table does not contain the computational time of sequence-template alignment.

6. Conclusion

In this paper, we propose a new machine learning approach—LS_Boost—to solve the protein fold recognition problem. We use a regression approach which is proved to be both more accurate and efficient than classification based approaches. One of the most significant conclusions of our experimental evaluation is that we do not need to calculate the standard z-score, and can thereby achieve a substantial computational savings without sacrificing prediction accuracy. Our algorithm achieves strong sensitivity results compared to other fold recognition

Table 5. Running time of different machine learning approaches.

	Training time	Testing time
LS_Boost	30 seconds	2 seconds
SVM classification	19 mins	26 mins
SVM regression	1 hour	4.3 hours
Neural Network	2.3 hours	2 mins
Naïve Bayes Classifier	1.8 hours	2 mins
Bayes Classifier	1.9 hours	2 mins

methods, including both machine learning methods and z-score based methods. Moreover, our approach is significantly more efficient for both the training and testing phases, which may allow genome-scale scale structure prediction.

References

1. T. Akutsu and S. Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 210:261–275, 1999.
2. N.N. Alexandrov. SARFing the PDB. *Protein Engineering*, 9:727–732, 1996.
3. S.H. Bryant and S.F. Altschul. Statistics of sequence-structure threading. *Current Opinions in Structural Biology*, 5:236–244, 1995.
4. Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.
5. J.H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annuals of Statistics*, 29(5), October 2001.
6. L. Holm and C. Sander. Decision support system for the evolutionary classification of protein structures. 5:140–146, 1997.
7. J. Moultand T. Hubbard, F. Fidelis, and J. Pedersen. Critical assessment of methods on protein structure prediction (CASP)-round III. *Proteins: Structure, Function and Genetics*, 37(S3):2–6, December 1999.
8. T. Joachims. *Making Large-scale SVM Learning Practical.* MIT Press, 1999.
9. D.T. Jones. GenTHREADER: An efficient and reliable protein fold recognition method for genomic sequences. *Journal of Molecular Biology*, 287:797–815, 1999.
10. D. Kim, D. Xu, J. Guo, K. Ellrott, and Y. Xu. PROSPECT II: Protein structure prediction method for genome-scale applications. *Protein Engineering*, 16(9):641–650, 2003.
11. H. Li, R. Helling, C. Tang, and N. Wingreen. Emergence of preferred structures in a simple model of protein folding. *Science*, 273:666–669, 1996.
12. E. Lindahl and A. Elofsson. Identification of related proteins on family, superfamily and fold level. *Journal of Molecular Biology*, 295:613–625, 2000.
13. D. Michie, D.J. Spiegelhalter, and C.C. Taylor. *Machine learning, neural and statistical classification,(edit collection).* Elllis Horwood, 1994.
14. J. Moult, F. Fidelis, A. Zemla, and T. Hubbard. Critical assessment of methods on protein structure prediction (CASP)-round IV. *Proteins: Structure, Function and Genetics*, 45(S5):2–7, December 2001.
15. J. Moult, F. Fidelis, A. Zemla, and T. Hubbard. Critical assessment of methods on protein structure prediction (CASP)-round V. *Proteins: Structure, Function and Genetics*, 53(S6):334–339, October 2003.
16. A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. SCOP:a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.
17. Judea Pearl. *probabilistic reasoning in intelligent system:Networks of plausible inference.* Springer, 1995.
18. J. Shi, T. Blundell, and K. Mizuguchi. FUGUE: Sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *Journal of Molecular Biology*, 310:243–257, 2001.
19. V.N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.
20. J. Xu. Protein fold recognition by predicted alignment accuracy. *IEEE Transactions on Computational Biology and Bioinformatics*, 2:157 – 165, 2005.
21. J. Xu, M. Li, D. Kim, and Y. Xu. RAPTOR: optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology*, 1(1):95–117, 2003.
22. J. Xu, M. Li, G. Lin, D. Kim, and Y. Xu. Protein threading by linear programming. pages 264–275, Hawaii, USA, 2003. Biocomputing: Proceedings of the 2003 Pacific Symposium.
23. Y. Xu, D. Xu, and V. Olman. A practical method for interpretation of threading scores: an application of neural networks. *Statistica Sinica Special Issue on Bioinformatics*, 12:159–177, 2002.