Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 08-002

Improved prediction of protein model quality

Kevin Deronne and George Karypis

January 22, 2008

Improved prediction of protein model quality

Kevin W DeRonne* and George Karypis

Department of Computer Science & Engineering, University of Minnesota,
Minneapolis, Minnesota, United States of America

Email: Kevin W DeRonne*- deronne@cs.umn.edu; George Karypis - karypis@cs.umn.edu;

*Corresponding author

Abstract

Background: Methods that can automatically assess the quality of computationally predicted protein structures are important, as they enable the selection of the most accurate structure from an ensemble of predictions. Assessment methods that determine the quality of a protein's structure by comparing it against the various structures predicted by different servers have been shown to outperform approaches that rely on the intrinsic characteristics of the structure itself.

Results: We developed an algorithm to estimate the quality of a predicted protein structure using a consensus approach. Our method uses LGA to align the structure in question to the structures for the same protein predicted by different servers and estimates the per-residue error by averaging the distances across these alignments. On a dataset containing 892,299 positions from 4,969 CASP7 submissions, our method achieved a root mean squared error (RMSE) of 6.69Å, which is significantly better than the 8.21Å achieved by the winning scheme in CASP7 for this problem (Pcons). We further improved these results to 6.51Å by developing a scheme that carefully selects which distances to average based on the predicted quality of the overall structure. We also examined the use of machine learning approaches to learn an appropriate aggregation scheme, which led to a simple weight learning approach achieving a 2.61Å RMSE on a reduced dataset.

Conclusions: Our results show that the use of LGA alignments and aggregation of raw distances is the primary reason for its performance advantage. In addition, our results show that beyond a binary inclusion/exclusion decision, learning from the data a set of weights by which the structures of the different servers can be aggregated can further improve performance.

Background

The problem of predicting protein structure from amino acid sequence has yet to be fully solved, and experimentally determining protein structures requires extensive human input. Due to the relative ease of determining amino acid sequences, and the utility of structural information, the problem has attracted much attention. As the accuracy of protein structure prediction algorithms has greatly improved over the last ten years [1, 2], the ability to precisely determine the quality of protein structure predictions has gained importance. In an attempt to motivate improvements in this area, the most recent session of CASP[1] included a model quality assessment category [3]. Given a putative structure, competitors were asked to submit a quality score between 0.0 and 1.0, or to assign an error estimate (in Å) to each residue of the structure. Twenty-eight groups submitted full structure quality estimates, and nine of those submitted per-residue error estimates.

Among the different methods proposed for solving the quality assessment problem, Pcons [4] has proved to be the most successful approach in predicting both complete structure quality and errors for individual residues. The underlying assumption of this approach is that, within the ensemble of predicted structures, recurring structures and structural motifs have an increased probability of being high-quality (i.e. close to the native state). Pcons determines the quality of a structure in two steps. First, it performs pairwise LGscore alignments [7] between the query structure and all structures in the ensemble. Second, S-scores [8] computed from these alignments are aggregated and used to produce a prediction.

In this paper we focus on improving the per-residue error estimates. We present an approach that is based on the same idea as Pcons, but which shows increased performance in assigning error estimates to residues in predicted models. This new technique alters both of the primary elements in the Pcons method. In place of LGscore alignments, LGA alignments are used, and rather than S-scores, raw distances are aggregated. We evaluate our methods on a dataset that contained 892,299 positions from 4,969 CASP7 submissions. Compared against the actual per-residue errors, our method achieves a root mean squared error (RMSE) of 6.69Å and a correlation coefficient of 0.81. This performance is significantly better than that achieved by Pcons on the same dataset, which is 8.21Å and 0.75, respectively (difference is

[1] http://predictioncenter.org/

statistically significant at $p \leq 0.0001$). We also present a scheme that carefully selects which distances to aggregate based on the predicted quality of the overall structure, which further reduces the per-residue errors to 6.51Å. In addition to this new method, we examine the use of a linear perceptron, standard regression, support vector regression and a simple weight learning technique to learn an appropriate aggregation scheme. Our extensive experimental evaluation demonstrates that the simple learning algorithm can be used to effectively improve the performance when trained appropriately.

Results and Discussion

Error assessment via LGA distance averaging

In this section we describe our new method for predicting per-residue error. Let X be the amino acid sequence of the query protein and let S_X be its predicted 3D structure. Let S_X^t be the true 3D structure for X and let $\{S_X^1, S_X^2, \ldots, S_X^k\}$ be the structures of X predicted by k different structure prediction methods. For each residue x_i of X, let $d_j(x_i)$ be the distance between the ith residue of S_X and the ith residue of S_X^j obtained after structurally superimposing S_X and S_X^j using the LGA program [6]. We will refer to $d_j(x_i)$ as the *x_i's distance between the query and the jth predictor*. The predicted distance $d(x_i)$ between residue x_i in S_X and S_X^t (the error estimate for position i) is given by

$$d(x_i) = \sum_{j=1}^{k} w_j(x_i) d_j(x_i), \tag{1}$$

where $w_j(x_i)$ is a weight associated with the jth predictor for residue x_i, and $\sum_j w_j(x_i) = 1.0$.

The key idea behind this per-residue error prediction approach is that each of the k structures over which it averages can be considered an expert's prediction for X's real structure. Thus, the per-residue error can be determined by a weighted average over the per-residue distances to the structure of each expert. The various $w_j(x_i)$ parameters control how the distances between the query and the predictors are weighted. A straightforward approach is to make all these weights equal (i.e., $w_j(x_i) = 1/k$), which corresponds to simple averaging. We call this approach LGA–Distance.

The above method differs from Pcons in two important ways. First, it uses LGA alignments as opposed to LGscore alignments. Second, it averages the raw distances as opposed to S-scores [7]. S-scores were developed as part of an improvement over root mean square deviation calculations for global structural comparison [8] (see the Methods section for how the S-score is calculated).

To understand the impact of these choices, we examine the performance obtained by four different schemes as shown in Table 1. The first two schemes (LGA–Distance and LGA–S-score) use LGA to compute

alignments, and average either raw distances or S-scores, whereas the second two schemes (LGscore–Distance and LGscore–S-score) use LGscore to compute alignments, again averaging either raw distances or S-scores. Using LGscore alignments and averaging S-scores is identical to the Pcons approach, and so we equivalently refer to Pcons as the LGscore–S-score scheme. Table 1 shows the results on three different datasets (FDS, LDS, and LDS$^-$), with LDS and LDS$^-$ being subsets of the FDS dataset (for a complete description of how these are constructed, see the Methods section).

As can be seen from this table, the LGA–Distance scheme outperforms the other three schemes on all three datasets. For example, on the FDS dataset, LGA–Distance achieves an RMSE of 6.69Å, which is significantly lower than those achieved by the other schemes (11.32Å for LGA–S-score, 7.73Å for LGscore–Distance, and 8.21Å for Pcons). Similar performance trends are observed in terms of the correlation coefficient. Comparing the performance of the schemes using distance-based averaging over those that average S-scores, we see that in general, the former lead to better results. A possible explanation for this is that S-scores were developed in the context of a global structural comparison rather than in the context of two residues. Finally, comparing the impact to the overall quality of the alignment methods (LGA and LGscore) and averaging schemes (distances and S-scores), we see that in general the former play a somewhat more important role. For instance, the gains achieved by LGA–Distance over LGscore–Distance are higher than those achieved by LGscore–Distance over LGscore–S-score.

Motivated by the positive results obtained by LGA–Distance, we explore two possibilities for improving this method. First, we examine if using a better subset of predictions can improve the results. Second, we look at using learning algorithms to estimate the various $w_j(x_i)$ parameters directly from the data.

Selecting predictions based on predictor quality

Given multiple predictions for a structure, some of the predictions will be better than others. If the quality of each of the predictions is known, a consensus predictor can be built that averages only those predictions above a certain quality. One measure of the quality of a predicted structure is its LGA score [6] to the true structure. Table 2 shows how the performance of the LGA–Distance scheme is affected by only including predictions whose quality is above a given threshold. The first column shows a minimum LGA_S required for a prediction to be used. Comparing values within a specific row of this table we see that limiting the set of predictions to only those with a score greater than a given threshold does help, provided that the threshold is sufficiently large. Note how the second row of the second column does not fit the general trend. If the distance between the query and the prediction is high, predictions with a low LGA score can

still accurately predict the distance between the query and the true structure.

However, as knowing the exact quality of a prediction requires knowing the true structure, an approximation for this quality must be used in a predictive algorithm. To estimate the quality of a predicted structure in the absence of the true structure, ProQ [9] may be used. ProQ uses features such as atom-atom contacts and secondary structure predictions to gauge structure quality. It has been shown to produce scores that correlate well with the LGA score to the true structure. Table 3 is analogous to Table 2, with MaxSub scores from ProQ substituted for LGA_S values [2]. Selecting predictions based upon a ProQ threshold produces an improvement over using all predictions, though the improvement is somewhat small, and the ProQ threshold must be sufficiently large. Table 4 shows the relative performance of the LGA–Distance scheme on filtered (selected predictions) and unfiltered (all predictions) data. Filtered datasets only include predictions with a ProQ score greater than 0.5, and positions with fewer than four such predictions are eliminated. These results show that using ProQ to select which predictors to average improves the per-residue prediction results in terms of RMSE by about 4.3% for the LDS dataset, and by about 17% for the LDS^- dataset.

An improved prediction strategy

The results reported in Table 4 suggest that a hybrid predictor relying upon averaging—but carefully selecting which predictors to include—can improve upon the LGA–Distance scheme. Our hybrid predictor splits the FDS dataset into three subsets, and predicts each part in a different manner. The first subset includes all positions from the LDS dataset. These positions are predicted using the LGA–Distance scheme with only the filtered predictions. Restricting the set of predictions this way changes the weights used in Equation 1 on a per-position basis (i.e., given n valid predictions for a position, each prediction will have a weight of $1/n$). The second set includes any remaining positions with at least 34 non-zero predictions. Averaging over the top 34 predictions based on ProQ score generates the prediction for this set. If a position has more than 34 non-zero predictions, the extra predictions are not used. Our experiments (results not shown) indicate that using this number yields the best balance between available positions and performance. The third subset contains the remaining positions, and the prediction is just the average over all available predictions. For the 892,299 positions available in this study, 40.16% are in the first set, 59.14% are in the second set, and 0.71% are in the third set. Predicting positions for each set separately as

[2] In Table 3, a minimum of four predictions above the threshold are required for a reported average value. We found that this requirement increases the reliability of the ProQ quality estimations.

described above, and evaluating all positions together achieves an RMSE of 6.51Å. Using the LGA–Distance scheme for all values in all cases has an RMSE of 6.69Å. For comparison, over all predictions in this set for which Pcons made a CASP7 prediction, Pcons achieves an RMSE of 8.21Å. Thus, the approach of carefully selecting positions to average performs quite well.

Learning how to weight predictors

We investigate the extent to which the weights $w_j(x_i)$ of Equation 1 can be learned from the data. Specifically, we focus on the simpler problem for which $w_j(x_i)$ depends only on the predictor j; that is, the prediction is given by

$$d(x_i) = \sum_{j=1}^{k} w_j d_j(x_i), \tag{2}$$

where w_j is a weight associated with predictor j.

We formulate the problem of learning the w_j weights as the following supervised learning problem. Given a set of training examples x_i, each described by the tupple $(d_t(x_i), \langle d_1(x_i), d_2(x_i), \ldots, d_k(x_i)\rangle)$, where $d_t(x_i)$ is the actual distance between the ith residue of S_X and the ith residue of X's true structure (obtained using CASP7's protocol), learn the set of w_j values such that

$$\sum_{x_i} \left(d_t(x_i) - d(x_i)\right)^2 \tag{3}$$

is minimized.

We use four different schemes to learn these weights: support vector regression (SVR), standard least-squares regression, a technique we call predictor boost, and a linear perceptron.

Table 5 compares the performance of these schemes on the LDS dataset. This table shows the results of predicting examples in both the training and the testing sets in order to illustrate the generalization ability of the different learning schemes (i.e., the extent to which a scheme does or does not over-fit the training data). From these results we see that the predictor boost and linear perceptron achieve the best results, outperforming the other two learning schemes. Moreover, the predictor boost and linear perceptron techniques outperform LGA–Distance and Pcons, and the difference between each learning/non-learning pair is statistically significant at $p \leq 0.0001$. Comparing the results obtained on the train and test datasets we see that SVR over-fits the data as its RMSE on the training set is much smaller than on the testing set. With the exception of the linear perceptron and the predictor boost schemes, these results of Table 5 show that the learning techniques do not outperform LGA–Distance. One explanation as to why the regression-based schemes do not perform well is that filtering the datasets as described above removes a

great deal of information from the available data. In the FDS set (before filtering) there are about 42 million values available, while in the LDS set (after filtering) only about 8 million remain. These missing values are assigned to zero, which may confuse the learning algorithms, as a zero value can mean two different things depending on its source. If the original $d_j(x_i)$ value was filtered out, then a zero is uninformative. However, if the zero represents a true $d_j(x_i)$ value of zero then the distance between S_X and S_X^j at position i is zero. This means that the two structures align perfectly at this position. In this case $d_j(x_i)$ should be treated as a zero, but in the former case $d_j(x_i)$ should simply be ignored. The next sections describe two methods for adjusting training data in order to compensate for the issue of $d_j(x_i)$ values of zero.

Results of filling missing dimensions

In order to eliminate the confusion due to missing values we developed a scheme by which the missing values were filled using estimates obtained from the other examples in the training set. Specifically, we use the average values from each predictor to fill the empty predictions. (See the section entitled "Filling in missing values" in the Methods section for a precise description of how this is done.) Table 6 shows the results from learning models to predict the data that were filled in this manner. These results show that, when applied to filled data, the regression techniques achieve better performance than when applied to the original (unfilled) data. In particular, the RMSE obtained by SVR improves from 3.92Å down to 2.93Å. The overall best results (RMSE of 2.76Å) are obtained by the standard regression technique. These results are better than those obtained by LGA–Distance and Pcons. Also, even though the absolute improvement achieved by the standard regression over LGA–Distance is small, the difference between them is statistically significant at $p \leq 0.0001$. Note that the values for LGA–Distance in Tables 5 and 6 are different because the former represents averaging over unfilled data, and the latter represents averaging over filled data.

Tailored training sets

As an alternate approach to artificial filling-in of values, we developed a method in which a custom training set is built for each test position encountered. Specifically, for a given test position, the training set contains only those examples that have at least the same set of predictors present as those in the test position. Values for w_j are learned from the training set, and these weights are used to classify the query position q according to Equation 2.

Table 7 shows the results from testing models built using such tailored data sets. These results were

obtained on the LDS$^-$ dataset and they are not directly comparable to those reported in Tables 5–6, which were obtained on the LDS dataset. These results show that the predictor boost scheme achieves the best results (RMSE of 2.61Å), which is considerably smaller than the RMSE values achieved by the other three weight-learning schemes as well as Pcons (RMSE of 4.30Å), and slightly smaller than LGA–Distance (RMSE of 2.63Å). The performance improvement against LGA–Distance is statistically significant at $p \leq 0.015$, and the improvement against the other schemes is statistically significant at $p \leq 0.0001$. Analyzing the performance of the other three weight-learning schemes we see that, as was the case with most of the previous results, they under-perform the LGA–Distance method (even though they do better than Pcons).

Within Table 7, we see that the errors on the testing sets are lower than those on the training sets. This might seem counter-intuitive at first, but the correlation coefficients are worse on the test sets than the train sets. Also, the standard deviations on the test sets are quite large, indicating that the performance is not uniform across the test cases. These discrepancies probably result because the coverage of the positions is different for the training and testing sets. For example, a position with few non-zero values may never be selected for a training set, but positions with many non-zero values will be repeatedly selected for training sets.

Conclusions

The results presented in this study reveal several interesting trends. First, using LGA alignments to obtain $d_j(x_i)$ values outperforms using LGscore alignments to obtain S-scores (the method employed by Pcons). This is most evident in the test results for the FDS dataset (Table 1) for which Pcons achieves an RMSE of 8.21Å with a correlation coefficient of 0.75, while the LGA–Distance scheme has an error of 6.69Å with a correlation of 0.81. Second, in terms of correlation coefficient, averaging raw distances tends to outperform averaging S-scores. For example, for the FDS dataset (Table 1), LGA–Distance and LGscore–Distance achieve correlation coefficients of 0.81 and 0.76, respectively, which are higher than the 0.70 and 0.75 correlation coefficients achieved by the corresponding LGA–S-score and LGscore–S-score schemes. Third, a hybrid predictor using intelligent averaging shows improved performance over all simple averaging techniques (Table 4). Fourth, supervised learning approaches based on a linear perceptron and predictor boost can be used to learn how to best weight the different predictors in order to improve the accuracy of the predictions. The gains achieved by these approaches depend on the specific formulation of the learning problem. Learning formulations that select training sets and build models tailored to the specific positions

being predicted achieve the best performance over equal-weighting approaches.

Methods

Dataset

The complete set of data used in this study, referred to as the *full dataset* (FDS), consists of the positions from 4,969 first submissions (labeled TS1) from all groups in CASP7, for a total of 892,299 positions. This dataset is derived from the original 5,198 first submissions to CASP7 by (i) eliminating 229 submissions for which ProQ was unable to produce a quality score and (ii) discarding the positions in which the LGA alignment of a submission against its true structure resulted in a distance greater than 50Å for that position. ProQ version 1.2 is used to produce MaxSub scores for these structures. PSI-PRED [10] predicted secondary structure is obtained using NCBI's nr database as of April of 2006, which contained 3,584,739 sequences. These predictions are used in ProQ calculations. The LGA program [6] is used to align each pair of structures with the options `-3 -ie -d:4.0 -o0 -sda`. These are the same options used by the CASP7 assessors.

To evaluate the various weight-learning algorithms, a subset of the FDS dataset is selected that contained the submissions for those proteins that had at least four submissions with a ProQ score above 0.5. This results in a set of 31 proteins, containing 600 submissions, and 304,686 positions. We will refer to this dataset as the *learning dataset* (LDS). These submissions are obtained from a total of 51 different CASP7 servers. The entire LDS dataset can be viewed as a $304,686 \times 51$ matrix, referred to as the *LDS matrix*. The rows of this matrix are obtained by concatenating the residues of the 600 submissions, while the columns correspond to the different servers from which these submissions were obtained. An (i, j) entry in the matrix represents the distance between a residue of the row submission to the corresponding residue in the column submission. These distances are derived after super-imposing the structures of the two submissions using LGA. If a particular server j did not submit a prediction for a protein X (or its submission had a ProQ score that was bellow 0.5), the (i, j) entries for all rows i corresponding to protein X will be empty. Also, even if a server provided a high-quality submission for protein X, some of its (i, j) entries can be missing because (i) the server did not provide predictions for these positions, or (ii) the positions are removed because their distance to the true structure was greater than 50Å.

Evaluating weight learning algorithms

We evaluate the three data-based approaches for learning the weights described in this paper (unfilled, filled, and tailored training sets) using the LDS dataset as follows.

The unfilled and filled approaches are evaluated using a leave-one-protein-out framework. A protein is selected, and all positions from this protein are assigned to the test set. All remaining examples are used as the training set. The whole process is repeated for each available protein. The difference between the unfilled and the filled approaches is that in the latter, the empty entries of the LDS matrix are filled using the method described in the next section.

The tailored training set approach is evaluated under a leave-one-position-out framework. A single position x_i serves as the test set and all positions from proteins other than X are searched to find the training set. The training set for position x_i corresponds to those rows of the LDS matrix whose non-empty columns are a superset of the non-empty columns of x_i's row. Only the values from the set of non-empty columns in x_i are used when training. This creates a training set with no missing values (i.e., the sub-matrix formed by the rows corresponding to the training set and the set of columns in the test position is completely filled). Note that it would seem that this approach will require learning 304,686 different models (one for each row of the LDS matrix). However, since the non-empty positions of the rows of the same protein will usually be the same (leading to the same training sets), the number of models that actually need to be learned is much smaller. Occasionally, rows from the same protein will differ, due to incomplete CASP7 predictions, but these cases are rare. To further reduce the number of models that need to be learned, we did not test those positions whose training sets could not be used for at least nine other positions (i.e., each model that we learned had to be used to test at least ten positions). This reduced the total number of models learned to only 485 and allowed us to test 200,696 positions. As the set of positions tested by the tailored training set approach is a subset of the LDS dataset, we will refer to it as the LDS$^-$ dataset.

Filling in missing values

The scheme that we used to fill in the missing values in the LDS matrix is based on similar techniques that were developed in the field of collaborative filtering [11–13]. An empty (i, j) position is filled by assigning to it a value that is obtained by averaging over the non-empty positions of column j, while taking into account the values along the rows in which these non-empty positions occur. Specifically, let D be the LDS matrix, let $\mu_i = (\sum_j D(i, j))/m_i$ be the mean value of the m_i non-zero entries of row i in D, let D' be the matrix obtained from D by subtracting from each non-empty (i, j) position its row average (i.e,

$D'(i,j) = D(i,j) - \mu_i$), and let $\mu_j = (\sum_i D'(i,j))/m_j$ be the mean value of the m_j non-zero entries of column j in D'. Then, an empty position (i,j) is assigned the value $D(i,j) = \mu_i + \mu_j$. Note that, in the case of filling a testing set, the values for μ_j from the training set are used to fill any missing values. The advantage of this method is that it accounts for differences in the underlying structural alignments, which is important because each of the alignments between a row prediction and a column prediction provides its own context. Subtracting μ_i from each row places all of the rows into a generalized context, and allows for the accurate computation of μ_j values. By subsequently adding μ_i to μ_j, the method provides an estimate of what the μ_j value should be in the context of the original row i.

Pcons

Pcons values are taken from the CASP7 website [3] with the exception of some incorrect values resulting from the bug noted in [3]. Corrected values were obtained from the Pcons authors and the reported performance here reflects the new numbers. Pcons uses LGscore-based structural alignments in place of the LGA minimization employed in this paper. Let $LG_j(x_i)$ be the distance between the ith residue of S_X and the ith residue of S_X^j obtained after structurally superimposing S_X and S_X^j using the LGscore algorithm [7]. Let $pc(x_i)$ be the Pcons prediction corresponding to $d(x_i)$ in Equation 1. Pcons uses the following three equations to produce a prediction.

$$S_j(x_i) = \frac{1}{1 + (\frac{LG_j(x_i)}{\sqrt{5}})^2} \quad (4)$$

$$S(x_i) = \frac{1}{k}\sum_{j=1}^{k} S_j(x_i) \quad (5)$$

$$pc(x_i) = \sqrt{5}\sqrt{\frac{1}{S(x_i)} - 1} \quad (6)$$

Note that equation 4 corresponds to the S-score for a position in an LGscore alignment.

Weight learning algorithms

Linear perceptron

One way of learning values for w_j in Equation 2 is to use Rosenblatt's linear perceptron classifier [17]. This is an online learning algorithm that iteratively updates a weight vector w for each training example x based on the difference between its actual and predicted values. Pseudo-code for our implementation of

[3] http://www.predictioncenter.org/casp7/Casp7.html

Algorithm 1 Learning Weight Vectors with the linear perceptron algorithm

Input: S: Number of Predictors.
m: Number of Training Samples.
N: Number of Training Iterations.
Output: w: Weight Vector.

1: $w \leftarrow 1/k$
2: **for** $n = 1$ to N **do**
3: **for** $x = 1$ to m **do**
4: $e_j \leftarrow |d_j(x) - d_t(x)| \ \forall_j$
5: $\phi_j \leftarrow 1/(e_j + \sum_j e_j/S) \ \forall_j$
6: $\phi \leftarrow \phi/\|\phi\|_1$
7: $\alpha \leftarrow |d(x) - d_t(x)|/m$
8: $w \leftarrow w + \alpha\phi$
9: $w \leftarrow w/\|w\|_1$
10: **end for**
11: **end for**
12: Return w

this algorithm is shown in Algorithm 1. For each position, the linear perceptron determines the error of each predictor (line 4). Each predictor is then assigned a weight that is inversely related to its error and the vector of these weights (ϕ) is scaled to sum to one (lines 5 and 6). Note that in line 5, the $\sum_j e_j/S$ factor is used to reduce the difference between lower and higher weights. We found that using this smoothing factor improves results. The learning rate α is updated based on the error of the prediction for each example ($d(x)$), as determined using Equation 2. In the case of the unfilled LDS dataset, we use $d(x)/\sum_j w_j$ as the prediction. This prevents the sparsity of the set from skewing the values learned for w. The vector ϕ becomes the update to w (line 8), and w is scaled to sum to one after processing each training example (line 9). The final weights are the values of w after five iterations over the training data, as a test (results not shown) showed a small difference between the weights from the fourth and fifth iterations. Note that this is a variation on a traditional linear perceptron, in which w is updated according to $w \leftarrow w + \alpha(real - predicted)d$. We use the variation shown in Algorithm 1 because it performs better for our problem (results not shown).

Support vector regression (SVR)

We used the SVMLight implementation [18] for support vector regression. Default values were used for the tube width and regularization parameters. The details of this regression technique have been described in detail elsewhere [19] and will not be covered here.

Predictor Boost

The predictor boost technique learns weights that are proportional to the number of times a predictor achieves the best performance. If two predictors tie for a given example, the weight is split between them. More formally, we build a frequency vector f, where f_j stores the number of times predictor j achieved the closest prediction to the true value. After updating f for all training examples, $f/\|f\|$ becomes the weight vector.

Standard regression

We use Matlab for standard regression. We also experimented with a constrained regression formulation. In this formulation, the weights w_j of the predictors must be positive and sum to one. This regression formulation could not learn an appropriate set of weights in the majority of cases, so the results are not included here.

Author's contributions

KWD and GK designed the methods, and experimental setup. KWD carried out the implementation of the various methods, and computational experiments. KWD wrote the manuscript under GK's technical supervision and mentoring. Both authors read and approved the final manuscript.

Acknowledgements

This work was supported by NSF ACI-0133464, IIS-0431135, NIH RLM008713A, and by the Digital Technology Center at the University of Minnesota.

References

1. Kryshtafovych A, Č Venclovas, Fidelis K, Moult J: **Progress over the first decade of CASP experiments**. *Proteins: Structure, Function, and Bioinformatics* 2005, **61**(S7):225–236.
2. A Kryshtafovych KF, Moult J: **Progress from CASP6 to CASP7**. *Proteins: Structure, Function, and Bioinformatics* 2007, **69**:194–207.
3. Cozzetto D, Kryshtafovych A, Ceriani M, Tramontano A: **Assessment of predictions in the model quality assessment category**. *Proteins: Structure, Function and Bioinformatics* 2007, **69(Suppl 8)**:175–183.
4. Wallner B, Elofsson A: **Prediction of global and local model quality in CASP7 using Pcons and ProQ**. *Proteins: Structure, Function, and Bioinformatics* 2007, **69**(S8):184–193.
5. Wallner B, Elofsson A: **Identification of correct regions in protein models using structural, alignment, and consensus information.** *Protein Science* 2006, **4**:900–913.
6. Zemla A: **LGA: a method for finding 3D similarities in protein structures**. *Nucleic Acids Research* 2003, **31**(13):3370–3374.

7. Cristobal S, Zemla A, Fischer D, Rychlewski L, Elofsson A: **A study of quality measured for protein threading models**. *BMC Bioinformatics* 2001, **2**(5).

8. Levitt M, Gerstein M: **A unified statistical framework for sequence comparison and structure comparison**. *Proceedings of National Academy of Science, USA* 1998, **95**:5913–5920.

9. Wallner B, Elofsson A: **Can correct protein models be identified?.** *Protein Science* 2003, **12**:1073–1086.

10. Jones DT: **Protein Secondary Structure Prediction Based on Position-specific Scoring Matricies**. *J. Mol. Biol.* 1999, **292**:195–202.

11. Sarwar B, Karypis G, Konstan J, Riedl J: **Analysis of Recommendation Algorithms for E-Commerce**. In *Proceedings of ACM E-Commerce* 2000.

12. Sarwar B, Karypis G, Konstan J, Riedl J: **Item-based Collaborative Filtering Recommendation Algorithms**. In *WWW10* 2001.

13. Deshpande M, Karypis G: **Item-Based Top-N Recommendation Algorithms**. *ACM Transactions on Information Systems* 2004, **22**:143–177.

14. Heger A, Holm L: **Picasso: generating a covering set of protein family profiles**. *Bioinformatics* 2001, **17**(3):272–279.

15. Mittelman D, Sadreyev R, Grishin N: **Probabilistic scoring measures for profile-profile comparison yield more accurate short seed alignments**. *Bioinformatics* 2003, **19**(12):1531–1539.

16. Rangwala H, Karypis G: **Profile Based Direct Kernels for Remote Homology Detection and Fold Recognition**. *Bioinformatics* 2005, **(in press)**.

17. Rosenblatt F: **The perceptron: A probabilistic model for information storage and organization in the brain**. *Psychological Review* 1958, **65**:386–407.

18. Joachims T: *Advances in Kernel Methods: Support Vector Learning*, MIT-Press 1999 chap. Making large-Scale SVM Learning Practical.

19. Vapnik V: *The Nature of Statistical Learning Theory*. New York: Springer Verlag 1995.

Tables

Table 1: Performance of prediction using LGA–Distance, LGA–S-score, LGscore–S-score (Pcons) and LGscore–Distance

	Correlation Coefficient			
Dataset	LGA–Distance	LGA–S-score	LGscore–Distance	LGscore–S-score (Pcons)
FDS	0.81	0.70	0.76	0.75
LDS	0.89	0.88	0.79	0.79
LDS$^-$	0.91	0.89	0.80	0.80
	RMSE			
Dataset	LGA–Distance	LGA–S-score	LGscore–Distance	LGscore–S-score (Pcons)
FDS	6.69	11.32	7.73	8.21
LDS	2.80	3.05	4.31	4.19
LDS$^-$	2.63	3.04	4.35	4.30

Values in this table represent the Pearson correlation and root mean squared error between the predicted and true per-residue distances. Pcons results are taken from CASP use all distances for the positions in these sets.

Table 2: Using the LGA–Distance scheme with structures scoring above a given minimum.

	Selected Predictors		All Predictors		
LGA_S Minimum	Corr. Coeff.	RMSE	Corr. Coeff.	RMSE	Positions Used
0	0.81	6.69	0.81	6.69	892299
10	0.76	7.83	0.81	6.68	891804
20	0.81	6.09	0.82	6.03	859232
30	0.83	5.28	0.83	5.40	782685
40	0.85	4.42	0.83	4.75	707801
50	0.86	3.85	0.86	3.91	619229
60	0.87	3.54	0.87	3.68	542344
70	0.92	2.62	0.90	3.00	356028
80	0.91	2.78	0.89	3.10	260007
90	0.92	2.57	0.90	2.94	166706

The numbers on the last column indicate the number of positions with at least one prediction greater than or equal to the specified minimum.

Table 3: Using the LGA–Distance method with structures scoring above a given ProQ score.

	Selected Predictors		All Predictors		
ProQ Minimum	Corr. Coeff.	RMSE	Corr. Coeff.	RMSE	Positions Used
0.10	0.80	6.74	0.81	6.69	892299
0.15	0.79	6.70	0.81	6.56	880232
0.20	0.80	6.07	0.81	5.85	830102
0.25	0.80	5.78	0.82	5.49	773544
0.30	0.85	4.56	0.84	4.83	703241
0.35	0.87	3.88	0.85	4.22	616047
0.40	0.89	3.36	0.87	3.72	558465
0.45	0.89	3.05	0.88	3.32	432303
0.50	0.89	2.79	0.89	2.93	346735
0.55	0.88	2.79	0.89	2.78	217862
0.60	0.86	3.10	0.87	3.01	117721
0.65	0.88	2.92	0.89	2.80	63898

The numbers on the last column indicate the number of positions with at least four predictions greater than or equal to the specified minimum.

Table 4: Performance of LGA–Distance on filtered versus unfiltered datasets

	Correlation Coefficient		RMSE	
Dataset	Filtered	Unfiltered	Filtered	Unfiltered
LDS	0.90	0.90	2.51	2.62
LDS^-	0.72	0.70	2.12	2.48

Results are not included for the FDS dataset, as by definition this dataset includes all values. Values in this table represent the Pearson correlation coefficient and root mean squared error between the predicted and true per-residue distances.

Table 5: Average correlation and RMSE for the LDS dataset. Positions in the average must have four predictions with a ProQ score greater than 0.5.

	Train			Test	
Technique	Corr. Coeff.	RMSE	Corr. Coeff.	RMSE
Support Vector Regression	0.89	2.85	0.78	3.92
Standard Regression	0.84	3.48	0.85	3.31
Linear Perceptron	0.90	2.80	0.90	2.76
Predictor Boost	0.90	2.78	0.90	2.75
LGA–Distance	0.89	2.82	0.89	2.80
Pcons	0.80	4.18	0.79	4.19

*Constrained regression fails to learn a model in most cases.

Table 6: Average correlation and RMSE for the LDS dataset with filled values. Positions in the average must have four predictions with a ProQ score greater than 0.5.

	Train			Test	
Technique	Corr. Coeff.	RMSE	Corr. Coeff.	RMSE
Support Vector Regression	0.91	2.56	0.88	2.93
Standard Regression	0.92	2.44	0.89	2.76
Linear Perceptron	0.89	2.81	0.89	2.79
Predictor Boost	0.89	2.81	0.89	2.79
LGA–Distance	0.89	2.81	0.89	2.79
Pcons	0.80	4.18	0.79	4.19

*Constrained regression fails to learn a model in most cases.

Table 7: Average correlation and RMSE for LDS^{-}. Positions in the average must have four predictions with a ProQ score greater than 0.5.

	Train		Test	
Technique	Corr. Coeff.	RMSE	Corr. Coeff.	RMSE
Support Vector Regression	0.89	2.92	0.90	2.75
Standard Regression	0.89	2.87	0.88	3.14
Linear Perceptron	0.88	3.17	0.91	2.63
Predictor Boost	0.88	3.17	0.91	2.61
LGA–Distance	0.88	3.18	0.91	2.63
Pcons	0.81	4.31	0.80	4.30