RMSD Approximation

COCS 174, Project Milestone
Milka Doktorova

May 8, 2012

1 Brief Overview

The goal of this project is to compute a function that approximates the RMSD (root mean square deviation) between two protein structures. Traditionally, the RMSD is computed from the 3-D atomic coordinates of the proteins after superimposing the structures. I want to find a function that approximates the RMSD from the structures' BAT (bond-angle-torsion) coordinates instead (thus circumventing the structural alignment step).

So far I have implemented two approaches: one based on kernel regression and one based on support vector (SV) regression. Following is a description of the data collection process, the two methods, their implementation and corresponding results.

2 Data Collection

The data consists of 1000 fragments extracted from a pool of 18,808 protein structures with sequence similarity less than 30%, downloaded from the Protein Data Bank (PDB). At most one fragment was extracted from a single structure.

Each fragment consists of 30 C^α atoms from consecutive residues in the protein backbone. The starting position of the extracted fragment from within the template protein structure was chosen at random. The first and last 3 residues of the backbone, as well as water molecules and other molecules bound to the protein were excluded from consideration.

All fragments were stored in PDB format, and used to compute the pair-wise RMSD scores, as well as each fragment's BAT coordinates. The BAT coordinates have the form:

$$(d_1, ..., d_{n-1}, \gamma_1, ..., \gamma_{n-2}, \phi_1, ..., \phi_{n-3})$$

where d_i is the distance between C_i^α and C_{i+1}^α; γ_i is the angle defined by atoms i through $i+2$; and ϕ_i is the dihedral angle defined by atoms i through $i+3$. Since each fragment is 30 residues long, its BAT representation is in $\mathbb{R}^{84}$.

3 Data Representation

Since the goal is to find a function $f : \mathbb{R}^N \to \mathbb{R}$ that approximates the RMSD between two structures, I consider two different representations of the input vector $x \in \mathbb{R}^N$ to the function f. Let $B_i, B_j \in \mathbb{R}^{84}$ be the BAT coordinates of fragments i and j accordingly. The two representations are as follows:

1. $x = B_i - B_j$, thus $x \in \mathbb{R}^{84}$, and

2. $x = [B_i^T \ B_j^T]^T$, thus $x \in \mathbb{R}^{168}$.

Therefore, my data $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$ consists of m different $(x^{(i)}, y^{(i)})$ pairs where $x^{(i)}$ is a vector either in $\mathbb{R}^{84}$ or $\mathbb{R}^{168}$, and $y^{(i)}$ is the corresponding RMSD score of the two structures encoded in $x^{(i)}$.

4 Kernel Regression

4.1 Method

The first approach that I tried is based on a metric learning for kernel regression method presented by K. Weinberger and G. Tesauro in 2007. The goal is to optimize the cumulative leave-one-out quadratic regression error of the training examples:

$$\mathcal{L} = \sum_i (y^{(i)} - \hat{y}^{(i)})^2 \qquad \text{where} \qquad \hat{y}^{(i)} = \frac{\sum_{j \neq i} y^{(j)} k_{ij}}{\sum_{j \neq i} k_{ij}}$$

and $k_{ij} = k(x^{(i)}, x^{(j)}) \geqslant 0$ is the Gaussian kernel function but without the constant factor, and $\sigma = 1$. The optimization is performed by learning a matrix A so that the distance between the i-th and j-th examples used in the kernel function can be expressed as

$$d(x^{(i)}, x^{(j)}) = ||A(x^{(i)} - x^{(j)})||^2.$$

The gradient of $\mathcal{L}$ with respect to A is then

$$\frac{\partial \mathcal{L}}{\partial A} = 4A \sum_i (\hat{y}^{(i)} - y^{(i)}) \sum_j (\hat{y}^{(j)} - y^{(j)}) k_{ij} x_{ij} x_{ij}^T \qquad \text{where} \qquad x_{ij} = x^{(i)} - x^{(j)}.$$

4.2 Implementation

I implemented the method in MATLAB but it failed to produce any results. In all runs that I did A was converging very fast while the value of $\mathcal{L}$ was increasing which didn't make any sense. There were numeric problems with the computation of $\hat{y}^{(i)}$ and I added `eps` to both the numerator and denominator in the formula. The main problem though, was that since the kernel function in the model is Gaussian and we fix σ to be 1, each k_{ij} came out as a very small number and that contributed to the fast convergence of A. I tried training the model with $\sigma > 1$ which naturally slowed down the convergence of A but again, $\mathcal{L}$ either didn't change over the iterations or became larger rather than smaller. This fact makes me think that the complete failure of the method is more likely to be a result of my misinterpretation of the paper than of the inability of kernel regression in general to train a model for RMSD approximation.

5 SV Regression

5.1 Method

The second approach that I tried is based on ϵ-SV regression as described in the tutorial by A. Smola and B. Schölkopf published in 2003. The goal is to find a function $f(x^{(i)})$ that has at most

ϵ deviation from the given target $y^{(i)}$ for all examples in the training set. The dual representation of the problem is as follows:

$$\text{maximize:} \qquad -\frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) < x^{(i)}, x^{(j)} > + \sum_{i=1}^{m} y^{(i)}(\alpha_i - \alpha_i^*) - \epsilon(\alpha_i + \alpha_i^*)$$

$$\text{where:} \qquad w = \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)x^{(i)}$$

$$\text{subject to:} \qquad \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0 \ , \qquad \alpha^{(*)} \in [0, C]$$

As discussed in the paper, $C = 1/(\lambda m)$ where λ is a regularization constant. After solving the quadratic program and finding the optimal α and α^* vectors, b is computed as follows:

$$b = \min_i (-\epsilon + y^{(i)} - < w, x^{(i)} >).$$

5.2 Implementation

I solved the quadratic program in MATLAB by using a trial version of CPLEX. In particular, I used the function *cplexqp(H,f,A,b,Aeq,beq,lb,ub)*. However, since this was a trial version, I was able to train my model with up to only 150 examples. I used 6 different values of λ: $\{10^{-12}, 10^{-6}, 10^{-3}, 10^{-1}, 10, 10^2\}$ and computed the 10-fold cross-validation scores in each of the 6 cases. I repeated the procedure for 3 different values of ϵ: $\{0.05, 0.5, 1\}$ and for each of the two different representations of the input data. The corresponding tests were performed on a set of 150 test cases with similar distribution of the RMSD values (Figure 5).

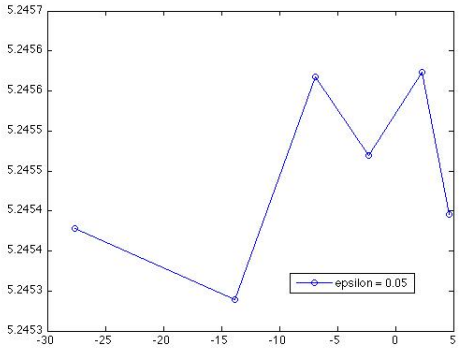
5.3 Results

Representation 1, $x^{(i)} \in \mathbb{R}^{84}$ Figure 1 shows the corresponding cross validation scores. In all three cases there is a very small variation in the mean squared error. Looking across the plots, the error increases as ϵ increases, which is expected since ϵ defines the error tolerance margins. However, the error still looks pretty large.

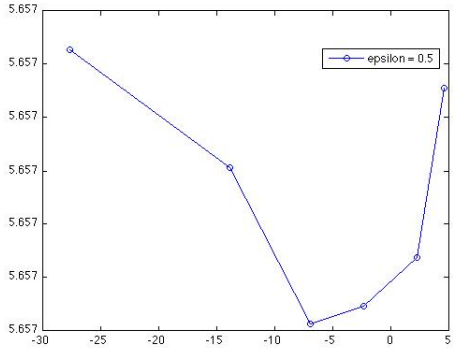
Figure 2 shows a comparison of the performance of the learned model over the training and test examples. I am not sure how to interpret this. The model has the same behavior with the training and test examples but the all-similar error values seem to indicate that I need to probably sample more λs in order to be able to evaluate how good the model is.

Representation 2, $x^{(i)} \in \mathbb{R}^{168}$ Figure 3 shows the corresponding cross validation scores. Again, in all three cases there is a very small variation in the mean squared error. Overall, the error is slightly smaller than the one from the corresponding plots in Figure 1 which may be a mere consequence of the increased dimension of the input vectors.

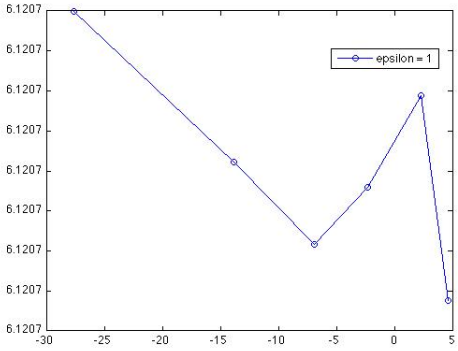
Figure 4 shows a comparison of the performance of the learned model over the training and test examples. These plots are almost identical to the ones in Figure 2 which indicates that the quality of the learned model is not affected by the particular representation of the input vectors.



(a) $\epsilon = 0.05$



(b) $\epsilon = 0.5$



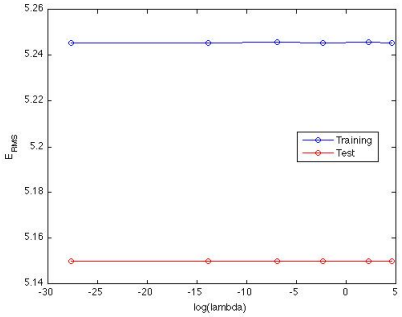
(c) $\epsilon = 1$

Figure 1: Representation 1: x-axis is $\log(\lambda)$, y-axis is square root of the cross validation score

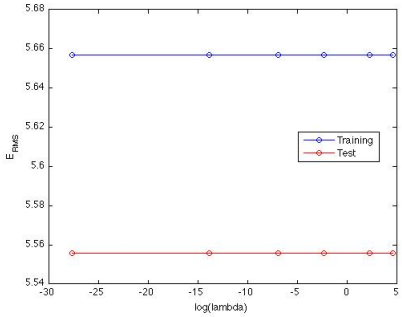
6 Discussion

The results so far are quite unsatisfactory. The errors produced by the trained models are very large and cannot be used to approximate RMSD within an acceptable error-tolerance range. I am curious to see if the quality of the model would change if I train it on a much larger training set. The full version of CPLEX should be able to handle that and it may be possible to obtain a free academic license. In addition, the model I implemented uses simply the dot product of the input vectors but it would be interesting to see what happens if a kernel function is used instead.

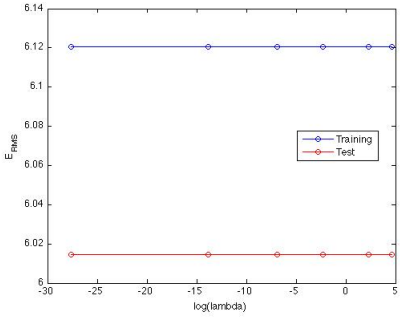
In general though, one such function wouldn't really yield an Euclidean metric for structure comparison that could be used for efficient nearest neighbor searches for instance. Consequently, an alternative direction for the project would be to design a model that produces a ranking of protein structures that corresponds to the ranking obtained by the structures' pair-wise RMSD scores. As of right now this seems like a more appropriate and feasible next direction than pursuing a better regression model.



(a) $\epsilon = 0.05$

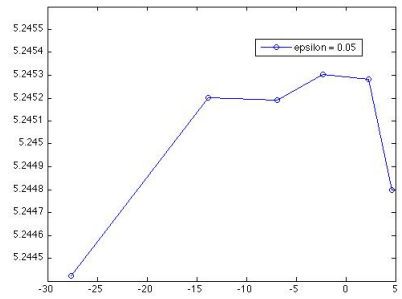


(b) $\epsilon = 0.5$

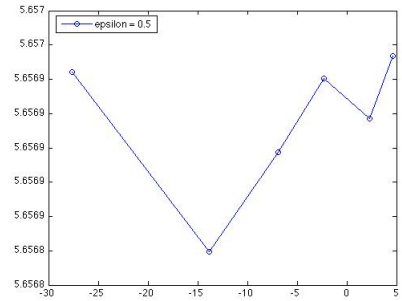


(c) $\epsilon = 1$

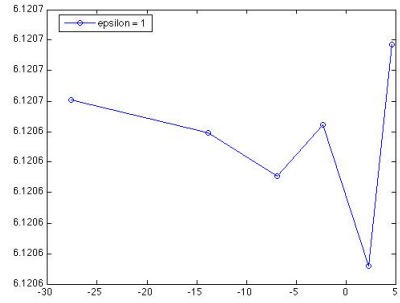
Figure 2: Representation 1: Testing the model on the training and test examples



(a) $\epsilon = 0.05$

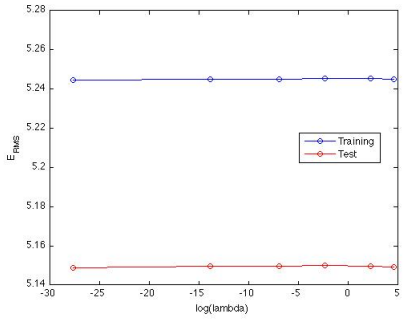


(b) $\epsilon = 0.5$

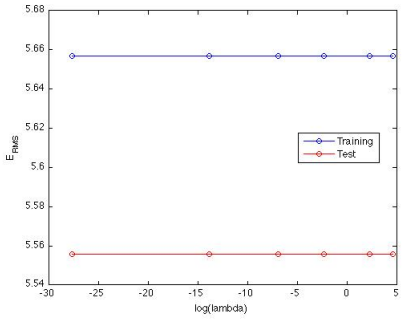


(c) $\epsilon = 1$

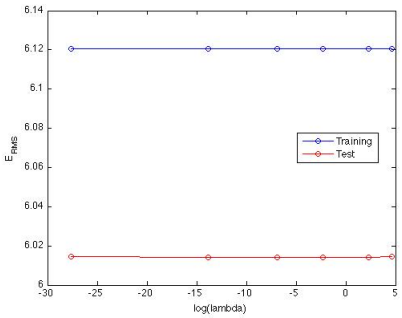
Figure 3: Representation 2: x-axis is $\log(\lambda)$, y-axis is square root of the cross validation score



(a) $\epsilon = 0.05$

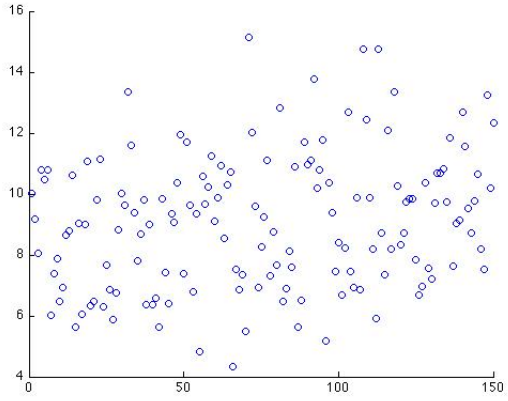


(b) $\epsilon = 0.5$

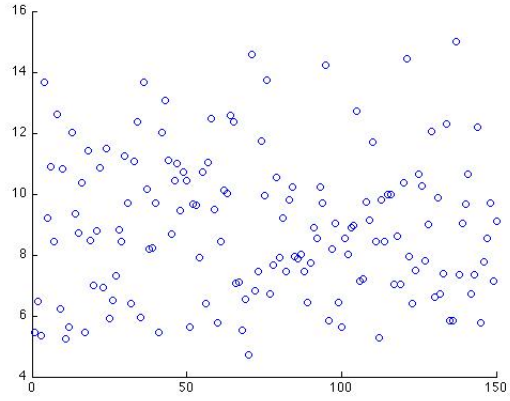


(c) $\epsilon = 1$

Figure 4: Representation 2: Testing the model on the training and test examples



(a) training examples



(b) test examples

Figure 5: Distribution of the RMSD values for the training and test examples. (y-axis is RMSD)