An Alternative to RMSD Computation

COCS 174, Final Project
Milka Doktorova

May 30, 2012

1 Introduction

Proteins are macromolecules composed of one or more chains of amino acids folded into compact 3-dimensional shapes. Their structure determines their function and consequently, the ability to determine structural similarity between proteins is important in structure-based analysis. A widely used metric for that is the root mean square deviation (RMSD) between the atomic coordinates of the two structures. However, RMSD computation requires initial alignment of the structures, which makes it expensive and cumbersome for use in large-scale studies.

In addition to the atomic coordinates however, a protein *conformation*, that is, the spacial arrangements of the atoms in the protein, can be represented in bond-angle-torsion (BAT) coordinates. The BAT coordinates consist of the bond distances and dihedral angles between the atoms in the molecule. As such, they fully determine the protein's structure but are independent of its particular embedding in $\mathbb{R}^3$. Therefore, the comparison of the BAT coordinates of two structures does not require the structures' initial alignment but simply computing the Euclidean distance in this representation does not yield an accurate measure of the similarity of the structures.

The initial goal of this project was to train a model to use the BAT coordinates in order to approximate the actual RMSD between them and Section 2 describes our attempts in this direction.

However, finding a good regression model turned out to be very hard due to many reasons. As a consequence, we decided to instead train a model that produces ranking of the structures similar to the ranking produced by RMSD. Section 3 describes our approach and results in this direction.

1.1 Data

Our 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^α_i and C^α_{i+1}; 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 $\mathbb{R}^{84}$.

Data for Regression Since the goal in the regression models is to find a function $f : \mathbb{R}^N \to \mathbb{R}$ that approximates the RMSD between two structures, we consider two different representations of the input vector $x \in \mathbb{R}^D$ 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, our data $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(n)}, y^{(n)})\}$ consists of n 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 $x^{(i)}$. For all the tests performed and discussed in Section 2 we use 150 training and 150 test examples that are independent of each other.

Data for Ranking When training a model on ranking the structures, we use the BAT coordinates of each training example directly. Thus, $x = \{x_1, x_2, ..., x_n\}$ with each $x_i \in \mathbb{R}^{84}$. For all the tests performed and discussed in Section 3 we use 100 training and 100 test examples that are independent of each other.

2 Regression

2.1 Kernel Regression

Method The first approach that we tried is based on a metric learning for kernel regression method presented by K. Weinberger and G. Tesauro [3]. 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

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

Implementation We implemented the method in MATLAB but it failed to produce any results. In all runs 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 we 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. We 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.

2.2 SV Regression

Method The second approach that we tried is based on ϵ-SV regression as described in the tutorial by A. Smola and B. Schölkopf [1]. 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:

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

Implementation We solved the quadratic program in MATLAB by using a trial version of CPLEX. In particular, we used the function *cplexqp(H,f,A,b,Aeq,beq,lb,ub)*. However, since this was a trial version, we were able to train a model with up to only 150 examples. We 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. This procedure was repeated 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).

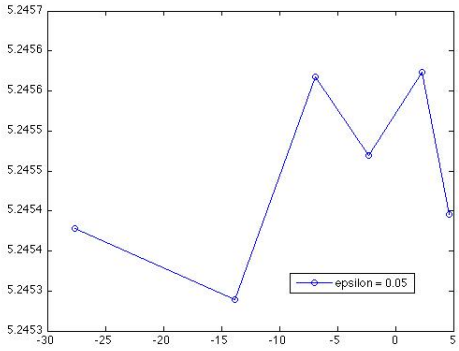
2.3 Results from SV Regression

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.

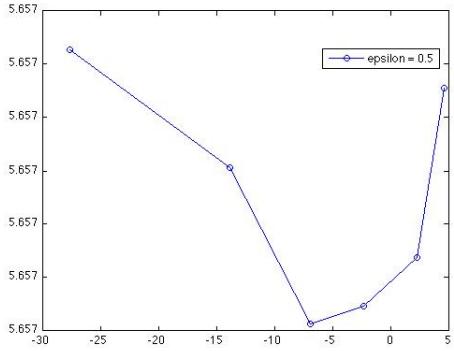
Figure 2 shows a comparison of the performance of the learned model over the training and test examples. The model has the same behavior with the training and test examples but the all-similar error values seem to indicate that we need to probably sample more λs in order to be able to evaluate how good 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.

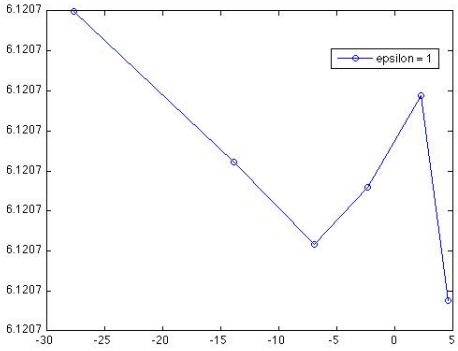
Figure 4 shows a comparison of the performance of the learned model over the training and test examples. These plots are almost identical to the ones in Figure 2 which indicates that the quality of the learned model is not affected by the particular representation of the input vectors.



(a) $\epsilon = 0.05$



(b) $\epsilon = 0.5$



(c) $\epsilon = 1$

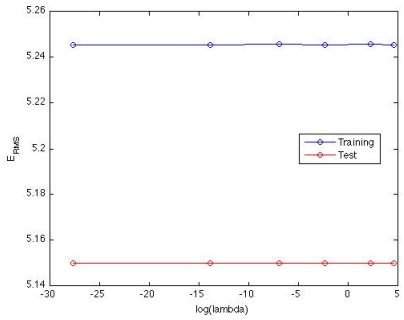
Figure 1: SV Regression, Representation 1: x-axis is $\log(\lambda)$, y-axis is square root of the cross validation score

2.4 Discussion

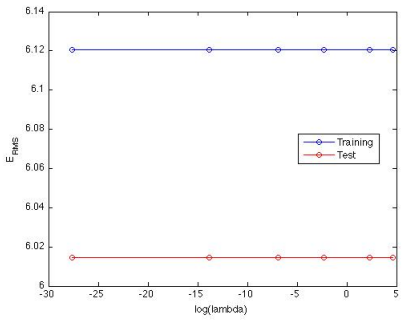
The results are quite unsatisfactory. The errors produced by the trained models are very large and cannot be used to approximate RMSD within an acceptable error-tolerance range. Possible directions for improving the model would include obtaining a full version of CIPLEX and training a model on a much larger training set, as well as incorporating a kernel function in the computation. In general though, this approach wouldn't really yield an Euclidean metric for structure comparison that could be used for efficient nearest neighbor searches for instance. Therefore, we decided to set a new goal for the project as described in the next section.

3 Ranking

We now want to train a model that produces a ranking of the structures similar to the ranking produced by RMSD. In order to do that we use a method developed by Torresani and Lee [2], called Large Margin Component Analysis (LMCA), and in particular its kernel version, kernel-LMCA. Our goal is to correctly identify the k-nearest neighbors of a protein without aligning the structures or computing RMSD, and instead, using only BAT coordinates.

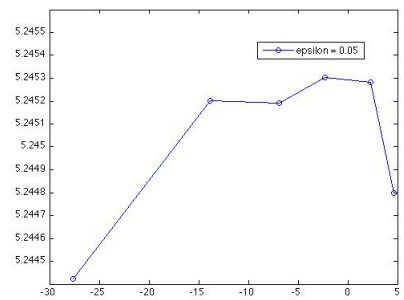


(a) $\epsilon = 0.05$

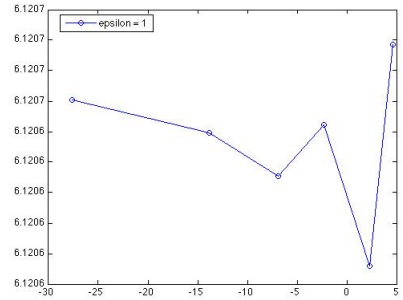


(b) $\epsilon = 1$

Figure 2: SV-Regression, Representation 1: Testing the model on the training and test examples



(a) $\epsilon = 0.05$



(b) $\epsilon = 1$

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

3.1 Method

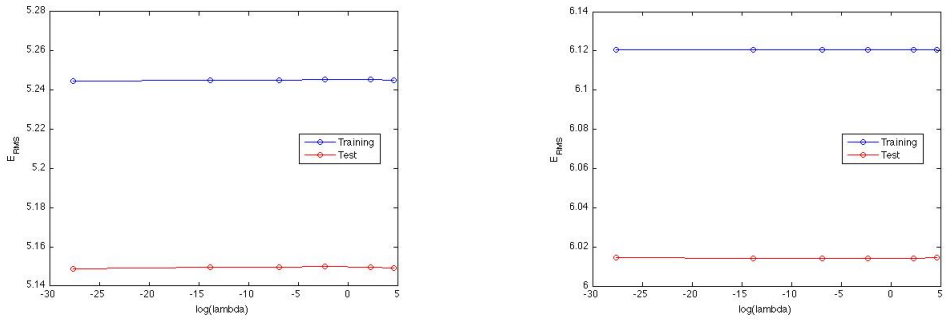
Consider input $x = [x_1, x_2, ..., x_n] \in \mathbb{R}^n$ with $x_i \in \mathbb{R}^D$. Kernel-LMCA computes non-linear features of the input vectors x that optimize the following learning objective:

$$\epsilon(L) = \sum_{ij} \eta_{ij} \|L(\phi_i - \phi_j)\|^2 + c \sum_{ijl} \eta_{ij} \upsilon_{il} h\left(\|L(\phi_i - \phi_j)\|^2 - \|L(\phi_i - \phi_l)\|^2 + 1\right).$$

Here η and υ can be 1 or 0: η_{ij} is 1 if x_j is one of the k-nearest neighbors of x_i RMSD-wise; υ_{ij} is 1 if x_j is not one of the k-nearest neighbors of x_i RMSD-wise but is one of the k-nearest neighbors of x_i BAT-wise, i.e. according to the Euclidean distance computed with the BAT coordinates. The function $h(s) = \max(s, 0)$ is a hinge function. The positive constant c denotes the relative importance of the two summation terms: the first one that pulls closer points that are neighbors RMSD-wise, and the second one that puts further apart points that are neighbors BAT-wise but not RMSD-wise (neighbors meaning within the k-nearest neighbors). The function $\phi : \mathbb{R}^D \to F$ maps input vectors to some high dimensional feature space F, and $\phi_i = \phi(x_i)$.

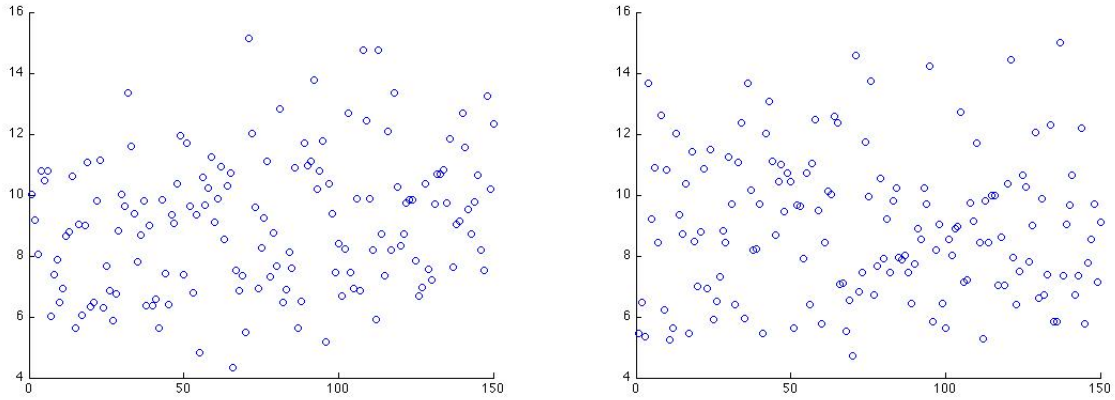
The learning objective $\epsilon(L)$ is minimized with a gradient descent procedure by setting $L = \Omega\Phi$ where $\Phi = [\phi_1, ..., \phi_n]$ and updating Ω with the following rule:

$$\Omega \leftarrow \Omega - \lambda\Gamma.$$



(a) $\epsilon = 0.05$ (b) $\epsilon = 1$

Figure 4: SV-Regression, 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 in SV-Regression. (y-axis is RMSD)

Here $\Omega, \Gamma \in \mathbb{R}^{d \times n}$ and Γ is as defined in the paper by Torresani and Lee. Upon convergence of Ω we project points using the kernel trick $L\phi_q = \Omega k_q$ where $k_q = [k(x_1, x_q), ..., k(x_n, x_q)]^T$ and $k(x, y) = \phi(x)^T \phi(y)$ is the kernel function.

3.2 Implementation

Initialization We implement the method in MATLAB using the SVM-KM Matlab package from `http://asi.insa-rouen.fr/enseignants/~arakotom/toolbox/`. For initialization of Ω we use the result from kernel Principal Component Analysis, k-PCA. We run k-PCA with different values of σ and for each of them, we select a different number d of eigenvectors corresponding to the first largest d eigenvalues. We choose σ and d for which the resulting Ω produces minimum classification error. The error for a single example is the fraction of the number of correctly identified k-nearest neighbors, and k. The classification error across all n examples then is the sum of the errors for the individual examples, divided by n. For our kernel function we use a Gaussian kernel:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right).$$

Parameters Following are the parameters of the model and the corresponding values that we have tested:

- k – number of nearest neighbors: $\{1, 3, 5, 7, 15, 20\}$
- σ – parameter for Gaussian kernel: $\left\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-1}, 10, 10^{2}, 10^{3}\right\}$
- d – number of eigenvectors, one of the dimensions of Ω: $\{1, 3, 5, ..., 99\}$
- λ – learning rate: $\{10^{-6}, 10^{-4}, 10^{-3}\}$
- c – relative importance of competing terms in learning objective: $\{0.6, 1.1, 1.6, 2.1, 3, 5, 10\}$

3.3 Results and Observations

Figure 6 shows the classification error evaluated on the training set using the ranking produced by RMSD and BAT coordinates (with Euclidean distance). First, we see that when $\lambda = 10^{-3}$ the model performs better than when λ is smaller. Setting $\lambda = 10^{-2}$ resulted in convergence problems, so we did not attempt to test larger values of λ. After initializing Ω with k-PCA, before training the model, the RMSD error in all three cases is almost identical to the classification RMSD error when $\lambda = 10^{-6}$ in which case the model converges immediately and our initial guess for Ω turns out to be the best reachable guess. Note that the classification RMSD error tends to decrease as k decreases which is expected since the number of nearest neighbors increases and the chances of error become smaller. Note also that the error from BAT coordinates after training the model doesn't have a well defined behavior and seems to increase as k increases which confirms that the model is not optimized to minimize the BAT error but the one coming from RMSD ranking.

We now fix λ to 10^{-3} and examine the effect of c on the classification error. Figure 7 shows the error evaluated using RMSD ranking. The initial error computed right after initialization is shown with a dashed black line. We see a clear trend indicating that larger values of c significantly reduce the error. Figure 8 shows the corresponding BAT error for the three largest values of c. Again, we see that it does not have a well defined behavior.

Finally, Figure 9 shows the error on the training and test sets for the values of λ and c found to be optimal from the tests above: 10^{-3} and 10 accordingly.

3.4 Discussion

Following are the optimal d and σ chosen during initialization of Ω for the different values of k:

k	1	3	5	7	15	20
d	7	11	39	81	61	89
σ	10^3	10^3	10^3	10^3	10^3	10^2

Note that when $k = 1$ we choose the fewest number of eigenvectors. From the graphs we see also that regardless of how we change the parameters the error for $k = 1$ always stays very high. This makes sense since it shows that correctly finding the single closest neighbor is very hard. Notice also that we don't test all possible values of d but only half of them. It is possible that there is a better optimal d that we have missed.

As a criteria for convergence in all test cases we use the norm of the difference between the old and new Ω and bound the number of iterations to 7. In the majority of the cases convergence was reached very fast and 7 seemed to be a reasonable number. For larger values of c though, more iterations may yield even better results.

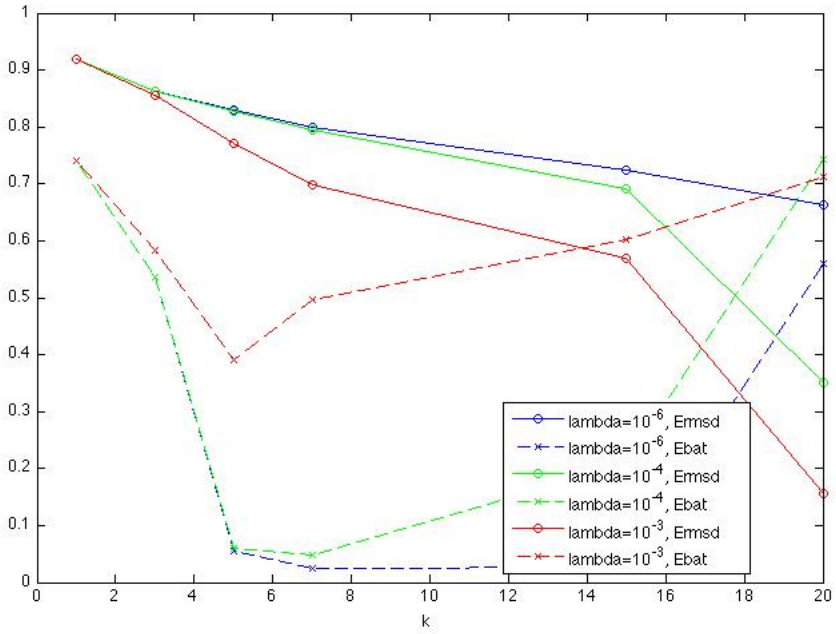


Figure 6: $[c = 1]$ Error of trained model evaluated on training set using RMSD ranking (Ermsd) and BAT ranking (Ebat) as a function of k. Initial error is almost identical to Ermsd for $\lambda = 10^{-6}$ and is not plotted.

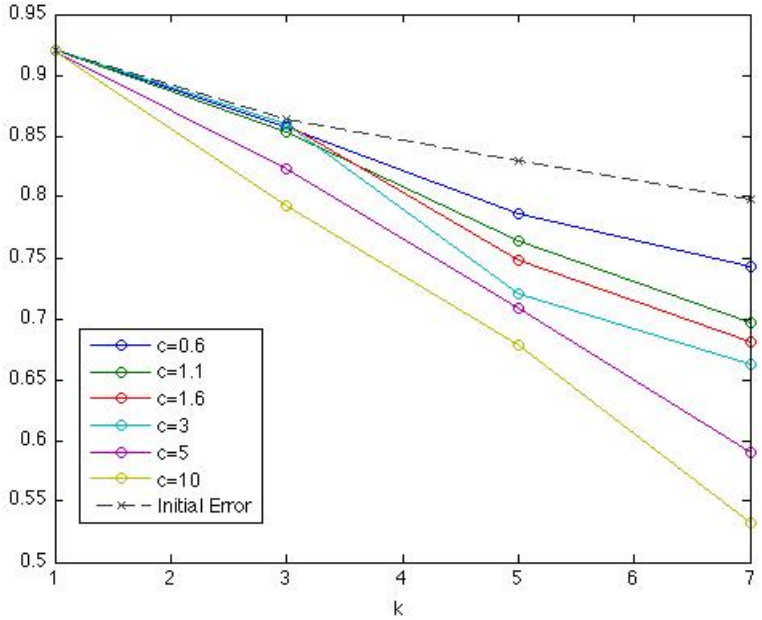


Figure 7: $[\lambda = 10^{-3}]$ Error of trained model evaluated on training set using RMSD ranking as a function of k.

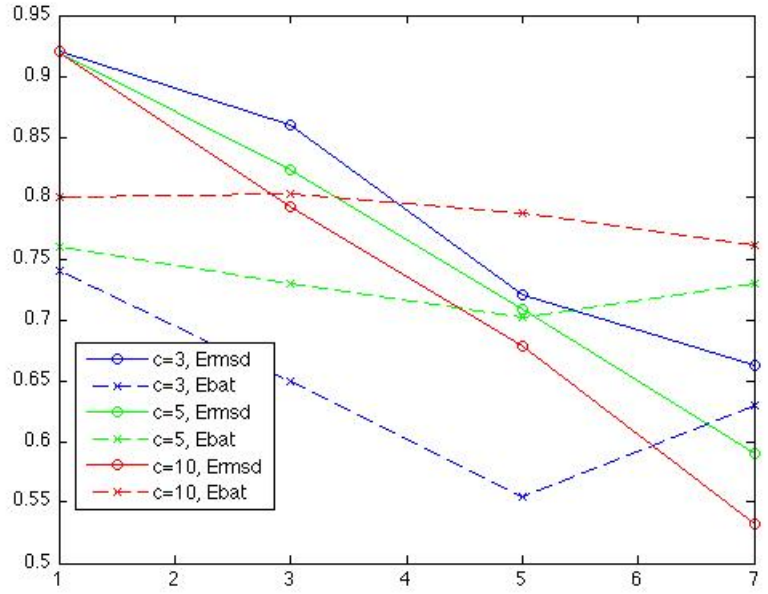


Figure 8: [$\lambda = 10^{-3}$] Error of trained model evaluated on training set using RMSD ranking (Ermsd) and BAT ranking (Ebat) as a function of k.

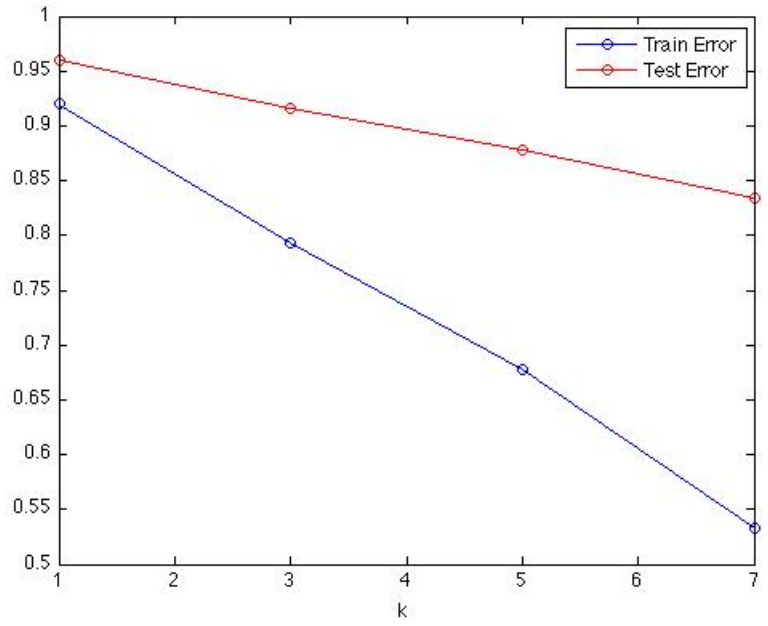


Figure 9: [$\lambda = 10^{-3}$, $c = 10$] Error of trained model evaluated on training and test sets using RMSD ranking as a function of k.

We tried different definitions of η and υ in order to better adopt the model to our problem but the ones listed here are the ones that produced best results.

Overall, varying so many parameters is challenging and requires careful analysis. Due to the high computational complexity resulting from the triply nested for loop and limited time frame, we were unable to perform more tests. However, the trends observed from our experiments give a good direction for more thorough analysis and indicate that it is possible to achieve better results. This can be done by training the model on a larger train set, trying all possible values for d, using $\lambda = 10^{-3}$, larger values of c and allowing for more convergence iterations.

References

[1] Smola and Schölkopf. A Tutorial on Support Vector Regression. Statistics and Computing, 2001

[2] Torresani and K.C. Lee. Large Margin Component Analysis. In B. Schölkopf, J. Platt, and T. Hofmann, editors, Advances in Neural Information Processing Systems 19, pp. 13851392. MIT Press, Cambridge, MA, 2007

[3] Weinberger and Tesauro. Metric Learning for Kernel Regression. Proceedings of International Workshop on Artificial Intelligence and Statistics, pp. 608615, 2007