Multi-class protein fold recognition using adaptive codes

Eugene Ie EIE@CS.COLUMBIA.EDU
Center for Computational Learning Systems, Columbia University, New York, NY 10115 USA

Jason Weston JASONW@NEC-LABS.COM
NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA

William Stafford Noble NOBLE@GS.WASHINGTON.EDU
Department of Genome Sciences, University of Washington, Seattle, WA 98195, USA

Christina Leslie CLESLIE@CS.COLUMBIA.EDU
Center for Computational Learning Systems, Columbia University, New York, NY 10115 USA

Abstract

We develop a novel multi-class classification method based on *output codes* for the problem of classifying a sequence of amino acids into one of many known protein structural classes, called *folds*. Our method learns relative weights between one-vs-all classifiers and encodes information about the protein structural hierarchy for multi-class prediction. Our code weighting approach significantly improves on the standard one-vs-all method for the fold recognition problem. In order to compare against widely used methods in protein sequence analysis, we also test nearest neighbor approaches based on the PSI-BLAST algorithm. Our code weight learning algorithm strongly outperforms these PSI-BLAST methods on every structure recognition problem we consider.

1. Introduction

Many statistical, homology-based methods have been developed for detecting protein structural classes from protein primary sequence information alone. They can be categorized into three major types of methods: pairwise sequence comparison algorithms (Altschul et al., 1990; Smith & Waterman, 1981), generative models for protein families (Krogh et al., 1994; Park et al., 1998), and discriminative classifiers (Jaakkola et al., 2000; Leslie et al., 2002; Liao & Noble, 2002). Many recent studies have shown that discriminative classifiers, such as support vector machines (SVMs), outperform the other two types of protein classification methods (Leslie et al., 2004) in the context of binary remote homology detection—prediction of whether a sequence belongs to a single structural class or not—especially when incorporating unlabeled protein data (Kuang et al., 2005; Weston et al., 2003). However, it is uncertain how to combine these predictive binary classifiers properly in order to tackle the multi-class problem of classifying protein sequences into one of many structural classes.

In the machine learning literature, two main strategies have been devised to tackle multi-class problems: formulating large multi-class optimization problems that generalize well-known binary optimization problems such as support vector machines (Vapnik, 1998; Weston & Watkins, 1999), or reducing multi-class problems to a set of binary classification problems and processing the binary predictions in simple ways to obtain a multi-class prediction (Allwein et al., 2000; Dietterich & Bakiri, 1995). The difficulty with the first method is that one usually ends up with a complex optimization problem that is computationally expensive. The second method is more computationally tractable, since it involves training a set of binary classifiers and assigns to each test example a vector of real-valued discriminant scores or binary prediction rule scores which we call the *output vector* for the example. In the standard one-vs-all approach, one trains N one-vs-the-rest classifiers to obtain a length N output vector, and one predicts the class with the largest discriminant score; standard all-vs-all is similar, but one trains all pairwise binary classifiers to obtain a length $N(N-1)/2$ output vector (Allwein et al., 2000). One can also represent different classes by binary vectors or *output codes* in the output vector space and pre-

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

dict the class based on which output code is closest to the binary output vector for the example (Dietterich & Bakiri, 1995; Crammer & Singer, 2000). (We will use the terms "output space" and "code space" interchangeably for the output vector space.) This approach, called error-correcting output codes (ECOC), appears more flexible than the other standard methods, though a recent empirical study suggests that the one-vs-the-rest approach performs well in most cases (Rifkin & Klautau, 2004). There has not been much work on applying newer multi-class techniques for protein classification, though recently, Tan et al. (2003) approached the multi-class fold recognition problem by statistically integrating one-vs-all and all-vs-all classifiers at the binary rule level to generate new classifiers for the final problem.

In this work, motivated by the strong performance shown by discriminative base classifiers for binary protein classification, we develop a multi-class approach for protein fold recognition based on output codes, where we integrate binary classifiers with extra information provided by a known hierarchical taxonomy, such as the manually curated Structural Classification of Proteins (SCOP) (Murzin et al., 1995). Our method somewhat resembles the error-correcting codes approach to multi-class classification. However, instead of using *ad hoc* output codes, we design codes that are directly related to the structural hierarchy, based on fold and superfamily detectors, and instead of using binary prediction scores, we solve an optimization problem to learn a *weighting* of the real-valued binary classifiers. Our approach is therefore conceptually related to the more general theory of adapting codes and embeddings developed by Ratsch et al. (2002), but we consider only the restricted problem of reweighting the output space so that our fixed codes perform well for the multi-class problem. To set up the optimization problem during training, we use a cross-validation scheme to embed protein sequences in output space by SVM discriminant score vectors, as described below.

In our experiments, we make use of two levels of the SCOP hierarchy as a method for designing codes for multi-class learning: *folds*, consisting of proteins with the same structural elements in the same arrangement; and *superfamilies*, subclasses of folds consisting of proteins with probable but remote shared evolutionary origin. Suppose the number of superfamilies and folds in a SCOP-based data set is k and q respectively. Our approach for solving the multi-class protein classification problem involves producing a real-valued output vector, $\vec{f}(x) = (f_1(x), ..., f_{k+q}(x))$, for each test sequence x, where the f_i are binary SVM superfamily or fold detectors trained using profile string kernels (Kuang et al., 2005), and using $(k+q)$-length binary code vectors $\mathbf{C}_j$ that combine superfamily and fold information. We use training data to learn a weight vector $\mathbf{W} = (W_1, \ldots, W_{k+q})$ to perform multi-class predictions with the weighted code prediction rule, $\hat{y} = \arg\max_j (\mathbf{W} * \vec{f}(x)) \cdot \mathbf{C}_j$, where $\mathbf{W} * \vec{f}(x)$ denotes component-wise multiplication. We learn $\mathbf{W}$ by a cross-validation set-up on the training set, using either a ranking perceptron or structured SVM algorithm. The full methodology consists of five steps: (1) split the training data into 10 cross-validation sets; (2) learn fold- and superfamily-level detectors from the partitioned training set—performing fold recognition and superfamily recognition on the held-out cross-validation sets, thereby generating training data for code weight learning; (3) use either the ranking perceptron algorithm or the structured SVM method for learning optimal weighting of classifiers in code space; (4) re-train superfamily and fold detectors on the full training set; and (5) test on the final untouched test set.

The rest of the paper is organized as follows. We first briefly provide an overview of how the base SVM classifiers using profile string kernels are constructed. Then we show how we incorporate structural hierarchy into codes, and we explain the theory of how to combine base classifiers through code embeddings using the ranking perceptron algorithm or structured SVM. This explanation is followed by the actual results on multi-class fold recognition, comparing our approach with three alternatives: an unweighted combination of maximum margin one-vs-all base classifiers, PSI-BLAST nearest neighbor searches on the non-redundant protein database, and Platt's conversion of SVM prediction scores to probabilities using sigmoid fitting (Platt, 1999) for direct classifier comparisons. Supplementary code and data can be found at `http://www.cs.columbia.edu/compbio/code-learning/`.

2. Profile-based string kernel SVM

For our base binary classifiers, we use profile-based string kernel SVMs (Kuang et al., 2005) that are trained to recognize SCOP fold and superfamily classes. The profile kernel is a function that measures the similarity of two protein sequence profiles based on their representation in a high-dimensional vector space indexed by all k-mers (k-length subsequences of amino acids). Specifically, for a sequence x and its sequence profile $P(x)$ (e.g. PSI-BLAST profile), the *positional mutation neighborhood* at position j and with threshold σ is defined to be the set of k-mers $\beta = b_1 b_2 \ldots b_k$ satisfying a likelihood inequality with respect to the corresponding block of the profile $P(x)$, as follows:

$$M_{(k,\sigma)}(P(x[j+1:j+k])) =$$
$$\{\beta = b_1 b_2 \ldots b_k : -\sum_{i=1}^{k} \log p_{j+i}(b_i) < \sigma\}.$$

Note that the emission probabilities, $p_{j+i}(b), i = 1 \ldots k$, come from the profile $P(x)$—for notational simplicity, we do not explicitly indicate the dependence on x. Typically, the profiles are estimated from close homologs found in a large sequence database; however, these estimates may be too restrictive for our purposes. Therefore, we smooth the estimates using the training set background frequencies $q(b)$, where $b \in \Sigma$ (the alphabet of amino acids), via $\tilde{p}_i(b) = \dfrac{p_i(b) + tq(b)}{1+t}$ for $i = 1 \ldots |x|$ and where t is a smoothing parameter. We use the smoothed emission probabilities $\tilde{p}_i(b)$ in place of $p_i(b)$ in defining the mutation neighborhoods.

We now define the profile feature mapping as

$$\Phi^{\text{Profile}}_{(k,\sigma)}(P(x)) = \sum_{j=0\ldots|x|-k} (\phi_\beta(P(x[j+1:j+k])))_{\beta\in\Sigma^k}$$

where the coordinate $\phi_\beta(P(x[j+1:j+k])) = 1$ if β belongs to the mutation neighborhood $M_{(k,\sigma)}(P(x[j+1:j+k]))$, and otherwise the coordinate is 0. Note that the profile kernel between two protein sequences is simply defined by the inner product of feature vectors:

$$K^{\text{Profile}}_{(k,\sigma)}(P(x), P(y)) = \langle \Phi^{\text{Profile}}_{(k,\sigma)}(P(x)), \Phi^{\text{Profile}}_{(k,\sigma)}(P(y)) \rangle.$$

The use of profile-based string kernels is an example of *semi-supervised learning,* since unlabeled data in the form of a large sequence database is used in the discrimination problem. Moreover, profile kernel values can be efficiently computed in time that scales linearly with input sequence length. Equipped with such a kernel mapping, one can use SVMs to perform binary protein classification on the fold level and superfamily level. We call these trained SVMs *fold detectors* and *superfamily detectors*.

3. Embedding base classifiers in code space

Suppose that we have trained q fold detectors. Then for a protein sequence, x, we form a prediction discriminant vector, $\vec{f}(x) = (f_1(x), ..., f_q(x))$. The simple one-vs all prediction rule for multi-class fold recognition is $\hat{y} = \arg\max_j f_j(x)$. The primary problem with this prediction rule is that the discriminant values produced by the different SVM classifiers are not necessarily comparable. While methods have been proposed to convert SVM discriminant scores into probabilistic outputs, for example using sigmoid fitting (Platt, 1999), in practice there may be insufficient data to estimate the sigmoid, or the fit may be poor. Our approach, in contrast, is to learn optimal weighting for a set of classifiers, thereby scaling their discriminant values and making them more readily comparable. We also incorporate information available from the superfamily detectors for doing multi-class fold recognition by designing *output codes* that encode information about the output space that is relevant to the structural class prediction problem.

We construct our codes to incorporate knowledge about the known structural hierarchy provided by SCOP (see Figure 2). Define for superfamily classes $j \in \{1, ..., k\}$, code vectors $\mathbf{C}_j = (\text{superfam}_j, \text{fold}_j)$, where superfam$_j$ and fold$_j$ are vectors with length equal to the number of known superfamilies (k) and folds (q), and each of these two vectors has exactly one non-zero component corresponding to structural class identity. Each component in $\mathbf{C}_j$ is known as a *code element* and represents the state of the corresponding classifier when we later combine the code elements to do multi-class prediction. For fold recognition, since every superfamily belongs to a fold in the SCOP hierarchy, each superfamily code also maps to a fold.

We adapt the coding system to fit the training data by learning a weighting of the code elements (or classifiers). The final multi-class prediction rule is $\hat{y} = \arg\max_j (\mathbf{W} * \vec{f}(x)) \cdot \mathbf{C}_j$, where $*$ denotes the component-wise multiplication between vectors. To learn the weight vector $\mathbf{W}$, we formulate a hard margin optimization problem as, $\min_{\mathbf{W}} ||\mathbf{W}||_2^2$, subject to $\left(\mathbf{W} * \vec{f}(x_i)\right) \cdot (\mathbf{C}_{y_i} - \mathbf{C}_j) \geq 1, \ \forall j \neq y_i$. Intuitively, our problem is to find an optimal re-weighting of the discriminant vector elements, such that a weighted embedding of the discriminant vector in code space $\mathbb{R}^{k+q}$ will exhibit large margins to discriminate between correct and incorrect codes (i.e. class identity).

We use two approaches to find approximate solutions to this optimization problem, the ranking perceptron and the structured SVM algorithm, as described below.

3.1. Using the ranking perceptron algorithm to learn code weights

The ranking perceptron algorithm (Collins & Duffy, 2002) is a variant of the well-known perceptron linear classifier (Rosenblatt, 1958). In our experiments, the ranking perceptron receives as input the discriminant vectors for training sequences and produces as output a weight vector $\mathbf{W}$ which is a linear combination of the input vectors projected onto code space. We modify the ranking perceptron algorithm such that it will learn our weight vector, $\mathbf{W}$, by satisfying n constraints:

$$\mathbf{W} \cdot (\vec{f}(x_i) * \mathbf{C}_{y_i} - \vec{f}(x_i) * \mathbf{C}_j) \geq m, \ \forall j \neq y_i \qquad (1)$$

where m is the size of the margin we enforce (Figure 1).

The update rule of the ranking perceptron algorithm can be different depending on what kind of loss function one is aiming to optimize. In standard *zero-one loss* (or classification loss), one counts all prediction mistakes equally, where $l_z(y, \hat{y})$ is 1 if $\hat{y} \neq y$, and 0 otherwise. The final zero-one empirical loss is $\frac{1}{n}\sum_i l_z(y_i, \hat{y_i})$. In *balanced loss*, the

(A) Learning code weights with the ranking perceptron algorithm

```
1: Define $F(x, y) = \mathbf{W} \cdot (\vec{f}(x) * \mathbf{C}_y)$
2: Input $\nu$:
3: $\mathbf{W} \leftarrow \vec{\mathbf{0}}$
4: for $i = 1$ to $n$ do
5:     $j = \arg\max_p F(x_i, p)$
6:     if $F(x_i, y_i) - m < \max_{q \in \{Y - j\}} F(x_i, q)$ then
7:         $\mathbf{W} \leftarrow \mathbf{W} + \nu k_i^{-1} \left( \vec{f}(x_i) * \mathbf{C}_{y_i} - \vec{f}(x_i) * \mathbf{C}_j \right)$
8:     end if
9: end for
10: Return $\mathbf{W}$
```

(B) Predicting examples with learned code weights

```
1: Define $F(x, y) = \mathbf{W} \cdot (\vec{f}(x) * \mathbf{C}_y)$
2: Input $\mathbf{W}$, $x_i$:
3: Return $\hat{y} \leftarrow \arg\max_j F(x_i, j)$
```

Figure 1. **Pseudocode for the ranking perceptron algorithm used to learn code weighting.** In the pseudocode, ν is the learning parameter; $k_i = |\{y_j : y_j = y_i\}|$ for balanced-loss, and $k_i = 1$, for zero-one loss.

cost of each mistake is inversely proportional to the true class size, where $l_b(y, \hat{y})$ is $\frac{1}{|y_i : y_i = y|}$ if $\hat{y} \neq y$ and 0 otherwise. The final balanced empirical loss is $\frac{1}{|Y|} \sum_i l_b(y_i, \hat{y}_i)$, where Y denotes the set of output labels.

Balanced loss is relevant to the protein structure prediction because class sizes are unbalanced, and we do not want to perform well only on the largest classes. The particular ranking perceptron training and prediction algorithms that we use are summarized in the pseudocode in Figure 1, including update rules for both zero-one and balanced loss.

3.2. Using structured SVMs to learn code weights

Support vector machines have been applied to problems with interdependent and structured output spaces in Tsochantaridis et al. (2004). These authors make use of a combined feature representation $\psi(x, y)$ as input vectors to learn a linear classification rule $\hat{y} = \arg\max_{y \in Y} \langle \mathbf{W}, \psi(x, y) \rangle$. Specifically, they use the $\psi(\cdot, \cdot)$ relation to discover input-output relations by forming $n|Y| - n$ linear constraints. These linear constraints specify that all correct input-output structures must be clearly separated from all incorrect input-output structures; i.e., for all i and all targets y different from y_i, we require that, $\langle \mathbf{W}, \delta\psi_i(y) \rangle > 0$, where $\delta\psi_i(y) \equiv \psi(x_i, y_i) - \psi(x_i, y)$. By defining, $\psi(x_i, y) = \vec{f}(x_i) * \mathbf{C}_y$, we arrive at linear constraints that are a special case of Equation 1. Using standard maximum-margin methods like SVMs, we obtain the hard margin problem

$$\min_{\mathbf{W}} \tfrac{1}{2} ||\mathbf{W}||_2^2$$
$$\forall i, \forall y \in \{Y - y_i\} : \langle \mathbf{W}, \delta\psi(y) \rangle \geq 1,$$

and the soft margin problem

$$\min_{\mathbf{W}, \xi} \tfrac{1}{2} ||\mathbf{W}||_2^2 + \tfrac{C}{n} \sum_{i=1}^{n} \xi_i$$
$$\forall i, \xi_i \geq 0; \forall i, \forall y \in \{Y - y_i\} : \langle \mathbf{W}, \delta\psi(y) \rangle \geq 1 - \xi_i,$$

where ξ_i corresponds to the slack variables (the amount an example can violate the margin), and C corresponds to the trade-off between maximizing the margin and the degree to which noisy examples are allowed to violate the margin. Intuitively, our definition of ψ defines the distance between two different protein embeddings in code space, and we are using large margin SVM methods to find the relative weighting of the dimensions in code space. Moreover, one can optimize balanced loss by rescaling the slack variables $\xi_i \leftarrow \dfrac{\xi_i}{l_b(y_i, y)}$ in the constraint inequalities.

4. Data set

4.1. Data set for training base classifiers

Our training and testing data is derived from the SCOP 1.65 protein database. We use ASTRAL (Brenner et al., 2000) to filter these sequences so that no two sequences share greater than 95% identity. Before running the experiments, we first remove all folds that contain less than three superfamilies so that there is a meaningful hierarchy in the remaining data. In addition, we select at least one superfamily from within the fold which contains more than one family, so that a *remote homology* detection problem can be properly constructed—that is, we want a superfamily recognition problem where the test sequences are only remotely related to (not from the same SCOP family as) the training sequences. Note that in some cases, there can be several superfamily recognition experiments within the same fold if there are enough multi-family superfamilies. Our data filtering scheme results in a data set that contains 46 SCOP superfamilies belonging to 24 unique SCOP folds. Details on training the binary superfamily and fold detectors are given immediately below.

4.1.1. Superfamily detectors

We completely hold out 46 SCOP families for final testing of multi-class prediction algorithms. In order to make our classification problem hard and meaningful, we find families that comprise at most 40% of the total number of sequences in their corresponding superfamilies, leaving only

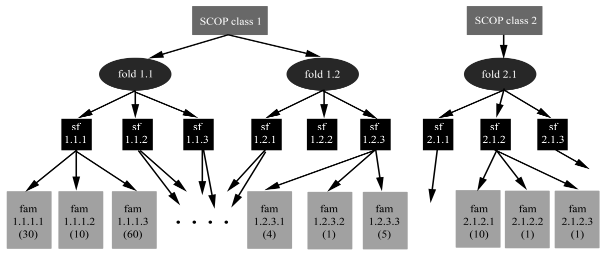

Figure 2. A hypothetical SCOP structure for illustrating data set construction, showing SCOP folds, superfamilies, and families. For example, the superfamily recognition problem for superfamily 1.1.1 will have $\{1.1.1.2, 1.1.1.3\}$ as positive training data, $\{1.1.1.1\}$ as positive testing data, $\{1.2.3.2, 1.2.3.3, 2.1.2.1, 2.1.2.3\}$ as negative training data, and $\{1.2.3.1, 2.1.2.2\}$ as negative testing data.

60% for learning. The important point to emphasize here is that the selected held-out families are completely untouched during any stages of training, so that a valid remote homology detection problem is in place. In general, our experimental setup is the same as the Jaakkola et al. (2000) remote homology detection experiment design, but we also ensure that the negative data for each superfamily detector does not include positive training data of other superfamily detectors in the same fold.

4.1.2. Fold recognition setup (fold detectors)

Fold recognition setup is analogous to the superfamily recognition setup, except that we now deal with recognition one level up the hierarchy. Furthermore, the sequences used in the superfamily setup are a proper subset of the fold setup.

4.2. Data set for learning weights

As outlined in the introduction, the first two stages of the method involve generating the appropriate training data for our weight learning algorithms. Specifically, we need vectors of SVM discriminant scores—given by trained fold and superfamily detectors—to embed sequences in output space in order to learn the code weighing. To do this, we use the following cross-validation scheme: we generate cross-validation experiments by randomly partitioning the positive training data of each superfamily (fold) detector into 10 mutually exclusive sets; in each cross-validation experiment, we train the superfamily (fold) detector in order to produce a set of discriminant vectors (SVM predictions) on the held-out set. Then we collect all the discriminant vectors from the 10 held-out sets to use as training data for our weight learning algorithms.

5. Methods

5.1. Baseline methods

Our main baseline comparison is the one-vs-all approach, using the 24 one-vs-all fold detectors directly for multi-class fold recognition with the prediction rule is $\hat{y} = \arg\max_j f_j(x_i)$. We also test one-vs-all with both fold and superfamily detectors; in this case, if a superfamily detector has largest discriminant score, our prediction is the fold that the superfamily belongs to.

To compare against a standard refinement of one-vs-all, we report the error rates achieved by applying Platt's sigmoid fitting (Platt, 1999) on each one-vs-all base classifier. We obtain the discriminants for learning the sigmoid function parameters by doing a 10-fold cross-validation on the training set.

We also compare our code weight learning approach to the multi-class performance of the widely used PSI-BLAST algorithm. We test two variants of a nearest neighbor approach using PSI-BLAST searches on the non-redundant protein database, called the query-based profile and target-based profile approaches. For the query-based profile approach, we build a profile around each test sequence and use the resulting profiles to rank all training sequences. We take the training sequence with the smallest E-value and predict its fold label for the query. For the target-based profile approach, we build a profile around each training sequence and use it to rank the list of test sequences, thereby computing an E-value for each test sequence. Then for each query, we use the training sequence that assigned the lowest E-value for our fold prediction.

Finally, to test our method on a previously published benchmark, we apply our code weight learning algorithm on an earlier multi-class protein fold recognition data set of Ding and Dubchak (2001) consisting of 27 SCOP folds, with 299 training examples and 383 test examples. More details on this data set are given in the results section.

5.2. Code weight learning methods

As base classifiers, we use the profile kernel SVM with profiles generated by PSI-BLAST on the non-redundant protein database. We perform cross-validation experiments which resulted in a total of 1772 training discriminant vectors for code weight learning. We use SPIDER to train our individual SVM detectors. SPIDER can be downloaded from `http://www.kyb.tuebingen.mpg.de/bs/people/spider/`.

5.2.1. Ranking perceptron

When learning code weights using the ranking perceptron algorithm, we first randomly permute the training data set and then run the perceptron algorithm iteratively until the squared norm of the difference between successive weight vectors is less than 0.01. For statistical robustness, we report results that average final error rates over 100 randomized runs. The learning parameter for all ranking perceptron experiments is set to 0.1, and the required margin is chosen to be $m = 2$.

5.2.2. Structured SVM

We implemented an instance of SVM-Struct (Tsochantaridis et al., 2004) and used it to learn code weights with the minimum constraint violation ϵ set at 10^{-8}. This variable controls the termination criterion of the SVM and thus directly affects the running time. Experiments with a subset of the data indicate that further increasing this termination threshold yields no significant improvements. The SVM margin-slack trade-off variable C is set at 0.1 for all structured SVM experiments.

6. Results

We report overall multi-class fold recognition results in Table 1 and Table 2. Table 3 shows the multi-class fold recognition results partitioned into biologically meaningful sets of SCOP folds. The error rate variances of reported averaged perceptron runs are in the order of 10^{-4} to 10^{-6}. We also note that whenever an experiment uses *full-length codes*, we employ a different measure of fold loss that is different from the simple "hard" rule of checking output label equality ($[\hat{y} = y]$). In the full-length codes scenario, a prediction is considered correct even if the label belongs to a different superfamily from the correct fold, i.e. we can simply infer the fold identity from the superfamily identity given the predefined hierarchical structure of SCOP.

Table 1. Multi-class fold recognition error rates using *full-length codes* via the ranking perceptron and an instance of SVM-Struct, compared to baseline methods. The results for the ranking perceptron algorithm are averaged over 100 randomized runs.

method	zero-one error	balanced error
sigmoid fitting	0.5592	0.7207
query-based PSI-BLAST	0.3728	0.4627
target-based PSI-BLAST	0.3195	0.4627
one-vs-all	0.1775	0.4869
SVM-Struct (optimize zero-one loss)	0.1627	0.2983
SVM-Struct (optimize balanced loss)	0.1361	0.3222
perceptron (optimize zero-one loss)	0.1592	0.3195
perceptron (optimize balanced loss)	0.1251	0.2795

Our results show that the code weight learning algorithm strongly outperforms Platt's algorithm on this particular problem, but we believe that the poor performance of sigmoid fitting is due to the small amount of training data available. We also outperform all PSI-BLAST nearest neighbor approaches and simple one-vs-all prediction rules. Our best approach uses *full-length codes* (superfamily and fold detectors) which yielded approximately 50% reduction in overall error rate for the multi-class fold recognition problem. Using *fold-only codes* (fold detectors only), we are also able to reduce error rates, but not as dramatically as full-length codes. When we look at the accuracy of the perceptron (trained using balanced loss) partitioned by different SCOP protein classes (α, β, α/β, $\alpha+\beta$, membrane proteins, small proteins), we outperform target-based PSI-BLAST method in balanced accuracy for 5 of 6 SCOP classes of folds, and all SCOP classes when compared against the query-based PSI-BLAST method.

Table 2. Multi-class fold recognition error rates using *fold-only codes* via the ranking perceptron and an instance of SVM-Struct, compared to baseline methods. The results for the ranking perceptron algorithm is averaged over 100 randomized runs.

method	zero-one error	balanced error
sigmoid fitting	0.5592	0.7207
query-based PSI-BLAST	0.3728	0.4627
target-based PSI-BLAST	0.3195	0.4627
one-vs-all	0.2189	0.5351
SVM-Struct (optimize zero-one loss)	0.1982	0.4717
SVM-Struct (optimize balanced loss)	0.2041	0.5024
perceptron (optimize zero-one loss)	0.1620	0.4073
perceptron (optimize balanced loss)	0.1749	0.4214

6.1. Use of Hierarchical Labels

We next investigated the nature of the strong improvement in accuracy of using full-length codes versus fold-only codes for multi-class fold classification. First, we retained all fold structures intact while randomizing superfamily structures within each fold to create a new data set with true fold structures but fictitious superfamily structures. Using this new perturbed data set, we repeated our experiments. The results, shown in Table 4, illustrate that the error rates obtained using these randomized superfamilies resemble the error rates obtained while using just fold-only codes. Looking at the weights returned from the code weights learning algorithms, we observed that the randomized superfamily full-length codes abandoned many superfamily detectors by setting their weights to 0. We also investigated using full-length codes with supervision at the fold level only, that is, only making updates in the perceptron or enforcing constraints in SVM-Struct based on fold labels rather than superfamily labels. Full-length codes with fold-level supervision outperformed fold-only codes but did not perform as well as full-length codes with full supervision. We conclude that the superfamily detectors and the hierarchical structure information incorporated into full-length codes play a significant role in reducing the error rates.

Table 3. A comparison of multi-class fold classification average accuracy using PSI-BLAST nearest neighbor method and using ranking perceptron codes learning algorithm with *full-length codes*. The ranking perceptron is trained using balanced loss.

SCOP class proteins	# ex.	method	zero-one accuracy	balanced accuracy
α (4 folds)	40	perceptron	1.0000	1.0000
		query-based PSI-BLAST	0.8500	0.8155
		target-based PSI-BLAST	0.8500	0.6984
β (9 folds)	145	perceptron	0.8552	0.7448
		query-based PSI-BLAST	0.6621	0.6264
		target-based PSI-BLAST	0.6483	0.4210
α/β (5 folds)	93	perceptron	0.8882	0.5789
		query-based PSI-BLAST	0.5806	0.4516
		target-based PSI-BLAST	0.8495	0.8277
$\alpha + \beta$ (3 folds)	18	perceptron	0.4611	0.4510
		query-based PSI-BLAST	0.3889	0.2879
		target-based PSI-BLAST	0.4444	0.3833
membrane (1 fold)	3	perceptron	1.0000	1.0000
		query-based PSI-BLAST	0.3333	0.3333
		target-based PSI-BLAST	0.0000	0.0000
small (2 folds)	39	perceptron	1.0000	1.0000
		query-based PSI-BLAST	0.5128	0.2703
		target-based PSI-BLAST	0.3846	0.4866

Table 4. Multi-class fold recognition using codes that incorporate different taxonomic information. The ranking perceptron algorithm is configured to optimize for balanced loss.

code type	balanced error
full-length	0.2795
full-length (randomized superfamilies)	0.3869
full-length (supervise only on folds)	0.4023
fold-only	0.4214

6.2. Comparison with Previous Benchmark

Finally, we compared our method to previously published multi-class results (Ding & Dubchak, 2001), which used physical-chemical features to represent protein sequences and one-vs-all and all-vs-all approaches with neural net and SVM classifiers. Since the original sequences were not retained, we extracted the current (possibly somewhat different) sequences from PDB corresponding to the given PDB identifiers. Relative to the current version of the SCOP hierarchy, 17% of the examples have mislabeled folds, but we keep the original labels for comparison purposes. We show in Table 5 that our base classification system strongly outperforms the previously published multi-class results, and despite the presence of mislabeled data, our code weight learning algorithm still manages to improve balanced accuracy by another 1%.

Table 5. Application of code weight learning algorithm on benchmark data set. SVM-2 is the previous benchmark's best classifier. The ranking perceptron is trained using balanced loss.

method	balanced accuracy
SVM-2	43.5%
one-vs-all (profile kernel)	72.5%
perceptron (profile kernel)	73.5%

7. Discussion

We have presented a novel method for performing protein fold recognition that significantly improves on directly using an ensemble of one-vs-all classifiers. We observe an average 50% reduction in overall balanced loss for the fold recognition problem. We also strongly outperform PSI-BLAST nearest neighbor methods on almost every class of fold (see Table 3). Both algorithms presented in Section 3.1 and Section 3.2 have similar performance, supporting the core idea of learning weighted codes that separates individual one-vs-all classifiers. Protein classification data is usually highly unbalanced, and a good learning algorithm must account for this imbalance in order to succeed. Both variants of the code learning algorithm use asymmetrical updates of constraint violations according to the observed relative class sizes. This approach strongly penalizes errors made in smaller sample populations, forcing the algorithm to concentrate on the less represented examples.

The presented approach is general and can be potentially applied to any problem with structure in the outputs. For the protein classification problem, our method can be naturally extended to adding further structure, and future work will investigate leveraging this additional information. For example, we can add family-based detectors to our chosen output space, or detectors from multiple algorithms, such as including both SVM and PSI-BLAST based detectors at once, and then learning their relative weights.

Acknowledgments

We would like to thank Thorsten Joachims for helpful suggestions on the implementation of SVM-Struct and Asa Ben-Hur for helpful comments on the manuscript. This work was supported by NSF grant EIA-0312706.

References

Allwein, E. L., Schapire, R. E., & Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. *Proc. 17th International Conf. on Machine Learning* (pp. 9–16). Morgan Kaufmann, San Francisco, CA.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). A basic local alignment search tool. *Journal of Molecular Biology*, *215*, 403–410.

Brenner, S. E., Koehl, P., & Levitt, M. (2000). The ASTRAL compendium for sequence and structure analysis. *Nucleic Acids Research*, *28*, 254–256.

Collins, M., & Duffy, N. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 263–270.

Crammer, K., & Singer, Y. (2000). On the learnability and design of output codes for multiclass problems. *Computational Learning Theory* (pp. 35–46).

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*, 263–286.

Ding, C. H., & Dubchak, I. (2001). Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, *17*, 349–358.

Jaakkola, T., Diekhans, M., & Haussler, D. (2000). A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, *7*, 95–114.

Krogh, A., Brown, M., Mian, I., Sjolander, K., & Haussler, D. (1994). Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, *235*, 1501–1531.

Kuang, R., Ie, E., Wang, K., Wang, K., Siddiqi, M., Freund, Y., & Leslie, C. (2005). Profile kernels for detecting remote protein homologs and discriminative motifs. *Journal of Bioinformatics and Computational Biology*. To appear.

Leslie, C., Eskin, E., Cohen, A., Weston, J., & Noble, W. (2004). Mismatch string kernels for discriminative protein classification. *Bioinformatics*, *20*, 467–476.

Leslie, C., Eskin, E., Weston, J., & Noble, W. S. (2002). Mismatch string kernels for SVM protein classification. *Advances in Neural Information Processing Systems 15*, 1441–1448.

Liao, C., & Noble, W. S. (2002). Combining pairwise sequence similarity and support vector machines for remote protein homology detection. *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology*, 225–232.

Murzin, A. G., Brenner, S. E., Hubbard, T., & Chothia, C. (1995). SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, *247*, 536–540.

Park, J., Karplus, K., Barrett, C., Hughey, R., Haussler, D., Hubbard, T., & Chothia, C. (1998). Sequence comparisons using multiple sequences detect twice as many remote homologues as pairwise methods. *Journal of Molecular Biology*, *284*, 1201–1210.

Platt, J. (1999). Probabilities for support vector machines. *Advances in Large Margin Classifiers*, 61–74.

Ratsch, G., Smola, A. J., & Mika, S. (2002). Adapting codes and embeddings for polychotomies. *Advances in Neural Information Processing Systems*, *15*, 513–520.

Rifkin, R., & Klautau, A. (2004). In defense of one-vs-all classification. *Journal Machine Learning Research*, *5*, 101–141.

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

Smith, T., & Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*, 195–197.

Tan, A., Gilbert, D., & Deville, Y. (2003). Multi-class protein fold classification using a new ensemble machine learning approach. *Genome Informatics*, *14*, 206–217.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support vector learning for interdependent and structured output spaces. *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 823–830.

Vapnik, V. N. (1998). *Statistical learning theory*. John Wiley and Sons, New York.

Weston, J., Leslie, C., Zhou, D., Elisseeff, A., & Noble, W. S. (2003). Cluster kernels for semi-supervised protein classification. *Advances in Neural Information Processing Systems 17*.

Weston, J., & Watkins, C. (1999). Support vector machines for multiclass pattern recognition. *Proceedings of the Seventh European Symposium On Artificial Neural Networks*.