Journal of Machine Learning Research () Submitted ; Published

Multi-class protein classification using adaptive codes

Iain Melvin* IAINMELVIN@GMAIL.COM
NEC Laboratories of America
Princeton, NJ 08540, USA; and
Center for Computational Learning Systems
Columbia University
New York, NY 10115, USA

Eugene Ie* TIE@UCSD.EDU
Department of Computer Science and Engineering
University of California
San Diego, CA 92093-0404, USA

Jason Weston JASONW@NEC-LABS.COM
NEC Laboratories of America
Princeton, NJ 08540, USA

William Stafford Noble NOBLE@GS.WASHINGTON.EDU
Department of Genome Sciences
Department of Computer Science and Engineering
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

Editor:

Abstract

Predicting a protein's structural class from its amino acid sequence is a fundamental problem in computational biology. Recent machine learning work in this domain has focused on developing new input space representations for protein sequences, i.e. string kernels, some of which give state-of-the-art performance for the binary prediction task of discriminating between one class and all the others. However, the underlying protein classification problem is in fact a huge multi-class problem, with over 1000 protein folds and even more structural subcategories organized into a hierarchy. To handle this challenging many-class problem while taking advantage of progress on the binary problem, we introduce an adaptive code approach in the output space of one-vs-the-rest prediction scores. Specifically, we use a ranking perceptron algorithm to learn a weighting of binary classifiers that improves multi-class prediction with respect to a fixed set of output codes. We use a cross-validation set-up to generate output vectors for training, and we define codes that capture information about the protein structural hierarchy. Our code weighting approach significantly improves on the standard one-vs-all method for two difficult multi-class protein classification problems: remote homology detection and fold recognition. Our algorithm also outperforms a previ-

*. The first two authors contributed equally to this work.

ous code learning approach due to Crammer and Singer, trained here using a perceptron, when the dimension of the code vectors is high and the number of classes is large. Finally, we compare against PSI-BLAST, one of the most widely used methods in protein sequence analysis, and find that our method strongly outperforms it on every structure classification problem that we consider.

Supplementary data and source code are available at `http://www.cs.columbia.edu/compbio/adaptive`.

Our adaptive code algorithm is used in a newly deployed protein fold recognition web server available called SVM-Fold, available at `http://svm-fold.c2b2.columbia.edu`.

Keywords: Multi-class classification, error-correcting output codes, structured outputs.

1. Introduction

Numerous statistical and supervised learning methods have been developed for detecting protein structural classes from primary sequence information alone. These methods can be categorized into three major types of approaches: pairwise sequence comparison algorithms (Altschul et al., 1990; Smith and 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., 2002b; Liao and Noble, 2002; Ben-Hur and Brutlag, 2003; Saigo et al., 2004). Many recent studies (see e.g. Leslie et al. (2004)) have shown that discriminative classifiers such as support vector machines (SVMs) used with appropriate sequence representations outperform the other two types of protein classification methods in the context of binary classification, i.e. prediction of whether a sequence belongs to a particular structural class or not. The binary classification performance of semi-supervised discriminative methods, which incorporate unlabeled protein sequence data into the learning algorithm, is particularly strong (Kuang et al., 2005; Weston et al., 2005). However, it is uncertain how best to leverage these accurate binary classifiers to solve the more important multi-class problem of classifying protein sequences into one of a vast number structural classes. Currently, for example, the manually curated Structural Classification of Proteins (SCOP, Murzin et al. (1995)) contains more than 1000 distinct 3D conformation classes called folds and even more structural subcategories (protein superfamilies and families). This complex prediction task provides a challenging problem for multi-class algorithms.

In the machine learning literature, two main strategies have been devised to tackle multi-class problems (reviewed in Rifkin and Klautau (2004)): formulating multi-class optimization problems that generalize binary classifiers like support vector machines (Vapnik, 1998; Weston and Watkins, 1999), or reducing multi-class problems to a set of binary classification problems and processing the output vectors of binary predictions to obtain a multi-class prediction (Allwein et al., 2000; Dietterich and Bakiri, 1995). The difficulty with the first method is that one usually ends up with a complex optimization problem that is computationally expensive. We therefore focus on the second, more computationally tractable approach, which encompasses standard methods like one-vs-all, all-vs-all, and error-correcting output codes. By "one-vs-all," we refer to the procedure of training N one-vs-the-rest real-valued classifiers to obtain a length-N output vector and testing new examples by predicting the class with the largest binary prediction score. All-vs-all is similar, except that one trains all pairwise binary classifiers to obtain a length $N(N-1)/2$ output vector (Allwein et al., 2000). In error-correcting output codes (ECOC), one rep-

resents different classes by binary vectors, called output codes, in the output vector space and predicts the class based on which output code is closest to the binary output vector for the example (Dietterich and Bakiri, 1995; Crammer and Singer, 2000). Despite the wide range of proposed multi-class solutions, a recent empirical study suggests that the simple one-vs-all approach performs as well or better than all other methods in most cases (Rifkin and Klautau, 2004).

One failing of one-vs-all is that it assumes that the prediction scores of the component binary classifiers are comparable, so that the individual classifier with the largest prediction corresponds to the best class. This assumption is often invalid in practice. One proposed remedy for SVM classifiers in particular is to fit a sigmoid function to the predicted margins for each classifier (Platt, 1999). After this procedure, the output probabilities rather than the margins are compared in one-vs-all. However, in many applications, the training data may be insufficient to fit the sigmoids accurately, or the sigmoids may be poor models for the margin distributions. Moreover, one-vs-all and the other standard output vector approaches do not take advantage of known relationships between classes, such as hierarchical relationships in the protein structural taxonomy, although there has been some recent work on hierarchical classification (Dekel et al., 2004; Cesa-Bianchi et al., 2006; Barutcuoglu et al., 2006). We further note that within the Bayesian learning community, alternative probabilistic strategies have been proposed for the multi-class problem, for example the multinomial probit model for multi-class Gaussian process classification (Girolami and Rogers, 2006).

In this work, we present a simple but effective multi-class method for protein structural classification that combines the predictions of state-of-the-art one-vs-the-rest SVM protein classifiers by supervised learning in the output space. In order to solve the problem that prediction scores from different classifiers are not on the same scale, we pose an optimization problem to learn a weighting of the real-valued binary classifiers that make up the components of the output vector. Instead of using *ad hoc* output codes as in ECOC, we design codes that are directly related to the structural hierarchy of a known taxonomy, such as SCOP, with components that correspond to fold, superfamily, and family detectors. We use a cross-validation set-up to generate output vectors as training data for learning weights, which we accomplish with a simple ranking perceptron approach. We note that Rätsch et al. (2002) considered a more general and difficult problem of adapting codes and embeddings, that is, learning both the code vectors and the embedding of the vector of prediction scores in output space via a non-convex optimization problem. In addition, Crammer and Singer (2000) formulated another more general problem of learning a mapping of all inputs to all outputs. By restricting ourselves to the simpler problem of reweighting the output space so that our fixed codes perform well, we are able to define a convex large-margin optimization problem that is tractable in very large-scale settings. We can also choose which loss function we wish to optimize. For example, in protein classification, we can use the balanced loss, so that performance on the large classes does not dominate the results.

The rest of the paper is organized as follows. In Section 2, we provide background on the protein classification problem, including our choice of base classifiers and construction of hierarchical codes. We then present our algorithmic approach for learning code weights in the output space using the ranking perceptron, describe different perceptron update rules, and compare to the code learning method of Crammer and Singer (2000) and other related work in Section 3. We provide large-scale experimental results on the multi-class remote

homology detection and fold recognition problems in Section 4, comparing our approach with a number of alternatives: standard one-vs-all, sigmoid fitting, PSI-BLAST (Altschul et al., 1997) used in a nearest neighbor approach to make multi-class predictions, and a perceptron version of Crammer and Singer's code learning method. We find that our adaptive code approach significantly outperforms one-vs-all in both multi-class problem settings and over all choices of code elements. We also strongly outperform PSI-BLAST for every structural classification problem that we consider. Finally, we find that our code learning algorithm obtains significantly better results than the higher capacity scheme of Crammer and Singer in the setting where the number of classes and dimension of the output space are both high. The current work is an expanded version of a conference proceedings paper (Ie et al., 2005). For this version, we have provided a much larger-scale experimental validation, added results on the fold recognition problem, introduced improved perceptron update rules and extended code vectors, and included a comparison with the Crammer and Singer method.

2. Background on protein classification: problems, representations, and codes

2.1 Remote homology detection and fold recognition

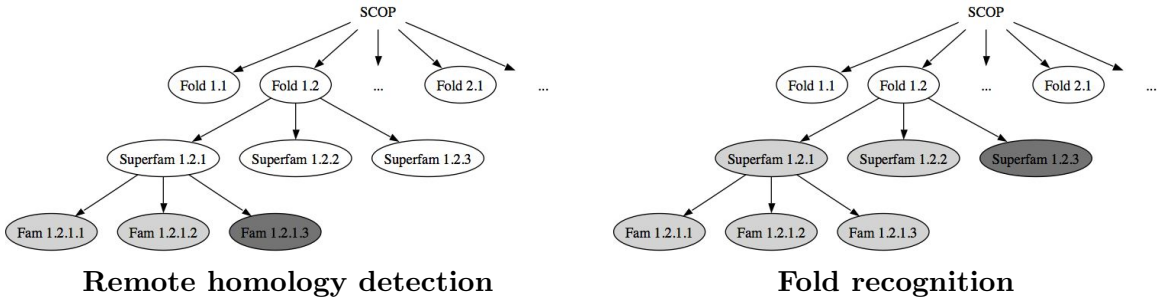

Figure 1: **Two protein classification problems.** (Left) In the SCOP database, we simulate the remote homology detection problem by holding out a test family (shown in dark gray) from a superfamily and using the other families as positive training data (shown in light gray). The task is to correctly predict the superfamily or fold membership of the held-out sequences. (Right) We simulate the fold recognition problem by holding out a test superfamily (dark gray) from a fold and using the other superfamilies as training data (light gray). The task is to correctly recognize the fold of the held-out sequences.

Protein classification is the prediction of a protein's structural class from its primary sequence of amino acids. This prediction problem is of fundamental importance in computational biology for a number reasons. First, a protein's structure is closely linked to its biological function, so knowledge of the structural category can allow improved prediction of function. Moreover, experimental methods for determining the full 3D structure of a protein

(X-ray crystallography, NMR) are time consuming and difficult and cannot keep pace with the rapid accumulation of unannotated protein sequences from newly sequenced genomes. Indeed, the complete repository of known protein structures, deposited in the Protein Data Bank, contains just 27K structures, while there are about 1.5M protein sequences in the Non-redundant Database of protein sequences. Second, prediction of a protein sequence's structural class enables the selection of a template structure from the database, which can then used with various comparative modeling techniques to predict a full 3D structure for the protein. Predicted structures are important for more detailed biochemical analysis and in particular for drug design. Note that template-based modeling approaches far outperform *ab initio* techniques for protein structure, i.e., methods that search for conformations that optimize an energy function without any template structure.

In this work, we focus on two protein classification problems that are considered unsolved in the structural biology community: remote homology detection and fold recognition. In remote homology detection, we wish to recognize when a new protein sequence has a distant evolutionary relationship to a protein sequence in a database (e.g., one whose structure is known). Due to a distant common ancestor, the protein sequences exhibit subtle sequence similarities (remote homology) that cannot generally be detected by statistical, alignment-based methods (Altschul et al., 1990, 1997). In fold recognition, we wish to recognize when a new protein sequence will exhibit the same fold as a protein from the structure database, even is there is no evidence of any evolutionary relationship between the proteins.

We base our experiments on SCOP, a manually curated hierarchical classification system for known protein structures. At the top level of the hierarchy are SCOP folds, consisting of sequences that have the same general 3D structural architecture. SCOP folds are divided into superfamilies, containing sequences that are at least remotely homologous (evolutionarily related). Each superfamily is further divided into families, consisting of homologous sequences with an easily detectable level of sequence similarity. We can design experiments based on the SCOP hierarchy to test performance on both the remote homology detection and the fold recognition problem, as depicted in Figure 1.

2.2 Profile-based fold, superfamily and family detectors

For our base binary classifiers, we use profile-based string kernel SVMs (Kuang et al., 2005) that are trained to recognize SCOP fold, superfamily, and family classes. We call these trained SVMs *fold detectors*, *superfamily detectors*, and *family detectors*. 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). A sequence profile is based on a multiple alignment of protein sequences to the input sequence and simply refers to the position-specific distribution of amino acids estimated from each column of the alignment. Intuitively, we use each k-length window of the sequence profile to define a *positional mutation neighborhood* of k-mers that satisfy a likelihood threshold, and the underlying feature map counts the k-mers from all the positional neighborhoods.

Specifically, for a sequence x and its sequence profile $P(x)$, the *positional mutation neighborhood* at position j and with threshold σ is defined to be the set of k-mers $\beta = b_1 b_2 \dots b_k$ 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.

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. The profile kernel between two protein sequences, i.e. the inner product of feature vectors, can be efficiently computed from the original pair of profiles using a trie data structure (Kuang et al., 2005).

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 (specifically, to estimate the probabilistic profiles).

A large variety of kernels have been designed specifically for protein sequences (e.g., (Jaakkola et al., 2000; Liao and Noble, 2002; Leslie et al., 2002a,b; Weston et al., 2005; Saigo et al., 2004; Ben-Hur and Brutlag, 2003; Rangwala and Karypis, 2005)). For this work, we selected the profile kernel because it is state-of-the-art. However, we have no reason to suspect that our primary conclusions regarding various multi-class classification methods depend on the choice of kernel function.

2.3 PSI-BLAST family detectors

PSI-BLAST (Altschul et al., 1997) is a widely used sequence comparison algorithm that builds a probabilistic profile around a query sequence, based on iterative alignment to database sequences. The resulting profile is then used to evaluate pairwise sequence similarities between the query and target sequences in the database. PSI-BLAST reports the significance of the similarity as an E-value (defined as the expected number of times that a similarity as strong or stronger than the observed similarity would be observed in a random protein sequence database of the given size), based on a profile-sequence alignment score. In theory, the E-value calculation makes PSI-BLAST results from different queries comparable to each other.

In this work, we use PSI-BLAST as a baseline method for comparison as well as a tool for generating sequence profiles for the profile kernel. In addition, we use PSI-BLAST to define an additional set of base classifiers for extended components of the output vectors in our multi-class approach.

For a given input sequence, we compute 1.0 - (the smallest PSI-BLAST E-value from the input sequence to training proteins in the family) and use this value as the component for the PSI-BLAST family detector in the discriminant vectors. Sequences with high similarity to one of the training sequences in the family receive a prediction score close to 1; if the E-value is highly insignificant, the score will be negative and large.

2.4 Output vectors and codes

To incorporate hierarchical labels into our output representation, we simply concatenate into a single output vector the one-vs-the-rest classification scores for classes at all relevant levels of the hierarchy. For example, in either the remote homology detection or fold recognition setting, both fold and superfamily detectors may be relevant for making fold-level predictions on test examples. Suppose the number of superfamilies and folds in a SCOP-based data set is k and q respectively. Then the real-valued output vector for each test sequence x would be $\vec{f}(x) = (f_1(x), ..., f_{k+q}(x))$, where the f_i are binary SVM superfamily or fold detectors trained using profile string kernels as described above. One can also extend the output vector to include binary family detectors. More generally, in this work we consider code elements that correspond to binary class detectors at the same level in the hierarchy as the target class (e.g. the fold level for fold predictions) as well as sublevels of the target class.

In the same output space, we define binary vectors that represent the hierarchical labels relevant for the problem. For the fold-level prediction example above, we define for superfamily classes $j \in \{1, ..., k\}$ the 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.

Our main idea is 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. In the next section, we describe how to learn $\mathbf{W}$ by using a ranking perceptron with a cross-validation set-up on the training set, and we develop update rules suited to the hierarchical problem.

3. Multi-class algorithms

3.1 Motivation: optimizing weights in output space

Given a fixed set of binary codes $\mathbf{C}_j$ and real-valued output vectors $\vec{f}(x)$ in the same output space $\mathbb{R}^N$ (see Section 2.4 for an example where $N = k + q$=#folds + #superfamilies), we want to adapt the coding system by learning a weight vector $\mathbf{W}$ so that the multi-class prediction rule $\hat{y} = \arg\max_j(\mathbf{W} * \vec{f}(x)) \cdot \mathbf{C}_j$ gives good empirical loss on the training data and generalizes well.

To learn $\mathbf{W}$, we first propose a hard margin optimization problem as

$$\min_{\mathbf{W}} ||\mathbf{W}||_2^2, \tag{1}$$

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 weighting of the output vector elements such that the re-weighted embedding of examples in the output space $\mathbb{R}^N$ will exhibit a large margin between correct and incorrect codes.

We use the ranking perceptron to find an approximate solution to this optimization problem, though a structured SVM approach (see Section 3.4.4) is also possible. Since

(A) Code weights learning

Define $F(x, y) = \mathbf{W} \cdot (\vec{f}(x) * \mathbf{C}_y)$
Input ν:
$\mathbf{W} \leftarrow \vec{\mathbf{0}}$
for $i = 1$ to n **do**
  $k = \arg\max_{p \in \{Y - y_i\}} F(x_i, p)$
  if $F(x_i, y_i) - m < F(x_i, k)$ **then**
    $\mathbf{W} \leftarrow \mathbf{W} + \nu n_i^{-1} \left(\vec{f}(x_i) * \mathbf{C}_{y_i} - \vec{f}(x_i) * \mathbf{C}_k \right)$
  end if
end for
Return $\mathbf{W}$

(B) Class prediction

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

Figure 2: **Pseudocode for the ranking perceptron algorithm used to learn code weighting.** In the pseudocode, ν is the learning parameter; $n_i = |\{y_j : y_j = y_i\}|$ for balanced-loss, and $n_i = 1$, for zero-one loss.

for the SVM base classifiers in particular, discriminant scores on training sequences are not very informative, we use a cross-validation set-up to produce prediction scores for the weight learning optimization. The full methodology consists of five steps: (1) split the training data into 10 cross-validation sets; (2) for each held-out fold, train a collection of fold-, superfamily-, and family-level detectors on the remaining data and use them to generate real-valued predictions on the held-out fold; (3) using the cross-validation scores to form output vectors, learn code weights with the ranking perceptron algorithm; (4) re-train fold, superfamily, and family detectors on the full training set; and (5) test on the final untouched test set.

3.2 Learning weights with the ranking perceptron

The ranking perceptron algorithm (Collins and 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 (generated through a cross-validation procedure) and produces as output a weight vector $\mathbf{W}$ which is a linear combination of the discriminant vectors projected onto the non-zero components of codes. We modify the ranking perceptron algorithm such that it will learn our weight vector $\mathbf{W}$ by satisfying n constraints:

$$\mathbf{W} \cdot (\vec{f}(x_i) * \mathbf{C}_{y_i} - \vec{f}(x_i) * \mathbf{C}_j) \geq m, \ \forall j \neq y_i, \tag{2}$$

where m is the size of the margin that we enforce (Figure 2).

The update rule of the ranking perceptron algorithm depends upon what loss function one is aiming to optimize. In standard *zero-one loss* (or classification loss), one counts all prediction mistakes equally,

$$l_z(y, \hat{y}) = \begin{cases} 1 & \text{if } \hat{y} \neq y; \\ 0 & \text{otherwise.} \end{cases}$$

The final zero-one empirical loss is $\frac{1}{n}\sum_i l_z(y_i, \hat{y_i})$. In *balanced loss*, the cost of each mistake is inversely proportional to the true class size,

$$l_b(y, \hat{y}) = \begin{cases} \frac{1}{|y_i : y_i = y|} & \text{if } \hat{y} \neq y; \\ 0 & \text{otherwise.} \end{cases}$$

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 2, including update rules for both zero-one and balanced loss.

3.3 The friend/foe and mean friend/foe update rules

When using codes representing multiple levels of the label hierarchy, we can also use relationships between codes to redefine the perceptron update rule. For a given label y, let friends(y) be the set of codes for classes belonging to the same superclass as y. For example, if we are using both superfamily and fold detectors for fold-level predictions and $C_y = (\text{superfam}_y, \text{fold}_y)$, the set of friends would be the codes for the superfamilies in the same fold as y (in particular, y itself belongs to friends(y)). We let foes(y) be all the codes that are not in friends(y). Then we can use the following update rule that updates all of friends(y) when the weakest friend does not beat the strongest foe by a margin:

1: $k = \arg\min_{p \in \{\text{friends}(y_i)\}} F(x_i, p)$
2: $l = \arg\max_{p \in \{\text{foes}(y_i)\}} F(x_i, p)$
3: **if** $F(x_k, \text{k}) - m < F(x_i, \text{l})$ **then**
4: $\quad \mathbf{W} \leftarrow \mathbf{W} + \nu n_i^{-1} \left(\vec{f}(x_i) * \mathbf{C}_{\text{friends}(y_i)} - \vec{f}(x_i) * \mathbf{C}_l \right)$
5: **end if**

In this rule, the vector $\mathbf{C}_{\text{friends(y)}}$ is the binary OR of all code vectors belonging to the superclass of y. We also implement a "mean friend/foe" rule whereby each element of $\mathbf{C}_{\text{friends(n)}}$ is the arithmetic mean of the occurrences of the code in all code vectors belonging to the superclass.

3.4 Comparison with existing approaches

3.4.1 The ranking perceptron and structured output learning

The ranking perceptron (Collins, 2000; Freund and Schapire, 1999) has been used to solve structured learning problems, particularly in natural language processing. Collins and Duffy (2002) trained parsers on the Penn Treebank that output a parse tree given a input sentence. This mapping from structured input to structured output is achieved by embedding both input and output objects into a joint feature space and computing for a test example:

$$\hat{y} = \arg\max_{y \in Y} \langle \mathbf{W}, \psi(x, y) \rangle.$$

Our approach follows the same framework, but we compute $\hat{y} = \arg\max_j (\mathbf{W} * \vec{f}(x)) \cdot \mathbf{C}_j$, where $\mathbf{W} * \vec{f}(x)$ denotes component-wise multiplication. That is, we choose the joint feature embedding:

$$\psi(x, y) = \vec{f}(x) * \mathbf{C}_y = \sum_i (\vec{f}(x))_i (C_y)_i$$

Our approach is thus an instance of structured output learning where the joint feature space captures dependencies between input and output variables. Firstly, inputs are modeled by $\vec{f}$ which uses classifiers relating to the levels of the label hierarchy in SCOP. Secondly, the outputs are modeled such that they are only dependent on classifiers from the relevant node or its ancestors.

3.4.2 Multi-class SVMs

Multi-class SVMs (Weston and Watkins, 1999; Vapnik, 1998) are a generalization of SVMs that handle the multi-class classification case by optimizing a single objective function. They use the rule $\hat{y} = \arg\max_{y \in Y} \langle w_i \cdot x \rangle$ as in the one-versus-all approach but enforce constraints of the form:

$$\langle w_{y_i} \cdot x \rangle - \langle w_y \cdot x \rangle > 1 \quad \forall y \in \mathcal{Y}, y \neq y_i.$$

Crammer and Singer (2002) later extended this approach by simplifying computations in the non-separable case (although the separable case remains the same). These constraints are identical to the ranking perceptron approach if, for that method, one selects the embedding (see Tsochantaridis et al. (2004)):

$$\psi(x, y) = \phi(x) \otimes \Lambda(y), \tag{3}$$

where $\Lambda(y) = (\delta_{1,y}, .., \delta_{N,y})^\top \in \{0, 1\}^K$, K is the number of clases, $\delta_{\cdot,\cdot}$ is the Kronecker delta, and $(a \otimes b)_{j+(i-1)K)} = a_i \cdot b_j$.

Multi-class SVMs are rather slow to train for non-linear systems which use kernels, since training in the dual has worst case complexity $O((mK)^3)$, where m is the number of training examples and K is the number of classes (Hsu and Lin, 2002). For this reason, we do not compare our approach to this method. However, our two-stage approach of training one-vs-all classifiers and then improving their performance with adaptive codes is tractable, because we train in primal variables and only have N input features, equal to the number of classifiers trained.

3.4.3 The Crammer-Singer method

Crammer and Singer (2000) also suggested a method for learning codes in a two-step fashion, which represents the closest algorithmic approach to ours that we know of. In their method, N weight vectors (one for each class) are optimized simultaneously but with no dependency between input and output features. The classification rule they use is $\hat{y} = \arg\max_i \mathbf{W_i} \cdot \vec{f}(x)$, and the feature space they use is identical to (3). In other words, they use the multi-class SVM to learn the second stage of their two-stage approach. This formulation means that the prediction of a given label i could depend on the output of any of the classifiers from the first stage, if the weight vector learned is not sparse. By contrast, in our approach, the ith label only depends on classifiers related to that label in the

1: Define $F(x, y) = \mathbf{W_y} \cdot (\vec{f}(x))$
2: $k = \arg\max_{p \in \{Y - y_i\}} F(x_i, p)$
3: **if** $F(x_i, y_i) - m < F(x_i, k)$ **then**
4: $\quad \mathbf{W_{y_i}} \leftarrow \mathbf{W_{y_i}} + \nu n_{y_i}^{-1} \vec{f}(x_i)$
5: $\quad \mathbf{W_k} \leftarrow \mathbf{W_k} - \nu n_j^{-1} \vec{f}(x_i)$
6: **end if**

Figure 3: Pseudo-code for the perceptron implementation of the Crammer-Singer code learning method detailed in Crammer and Singer (2000).

label hierarchy. In the case of a flat hierarchy (pure multi-class classification), our approach only rescales the classifiers in the one-vs-all approach, whereas the Crammer-Singer method learns, for each output, a weighted combination from all the one-vs-all classifiers. That is, they learn the more difficult problem of a mapping of all inputs to all outputs. Because of this, we hypothesise that the Crammer-Singer approach is likely to fail in the case of a large number of classes, as uncorrelated classes are essentially noisy features in the second stage of learning.

In Section 4 we make a comparison between our method and the Crammer-Singer approach. To facilitate comparisons, we implemented a perceptron-style learning of their algorithm (Figure 3), both with and without balanced loss. In our experiments, which use a very large number of classes, our approach indeed does outperform the one of Crammer and Singer.

3.4.4 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 input-output feature representation $\psi(x, y)$ as training 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,

$$\langle \mathbf{W}, \delta\psi_i(y) \rangle > 0 \quad \forall i, y \neq y_i,$$

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 2. Using standard maximum-margin methods like SVMs, we obtain the hard margin problem described by (1) above and the soft margin problem

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

where the ξ_i correspond to 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 the balanced loss by rescaling the slack variables $\xi_i \leftarrow \frac{\xi_i}{l_b(y_i, y)}$ in the constraint inequalities. However, in preliminary results (Ie et al., 2005), we found that the structured SVM gave similar performance to the ranking perceptron when used with our joint input-output embedding ψ, so we focus on perceptron approaches in the current study.

3.4.5 Loss functions and Energy Based Learning

Another general approach to structured output learning, called Energy Based Learning (EBL), was suggested by LeCun and Huang (2005), which is derived from the earliest approach to structured output prediction that we know of (Bottou et al., 1997). In EBL, for a given input, one chooses the output with the lowest energy:

$$\hat{y} = \arg\min_{y \in \mathcal{Y}} E(x, y).$$

One therefore seeks to use a loss function that pushes "down" the energy of the correct output(s) and pushes "up" the energy of other outputs. The authors show how different choices of loss function lead to different existing algorithms, including the ranking perceptron, which only pushes up the incorrect answers produced by the model, and negative log-likelihood, which pushes up the energies for all the examples with a force proportional to the likelihood of each answer under the model. Our friend/foe update rule can be seen in this framework as a different loss function that takes account of multiple output values that all give the same original loss.

3.4.6 Rätsch et al's Adaptive code learning

Rätsch et al. (2002) also considered the problem of adaptive code learning and proposed a general approach consisting of learning both code vectors and the embedding of the vector of prediction scores in output space. Their algorithm involves iteration between learning the codes and learning the embedding, resulting in a difficult non-convex optimization problem.

By restricting ourselves to the simpler problem of reweighting the output space so that our fixed codes perform well, we are able to define a convex large-margin optimization problem that is tractable in very large scale settings. Furthermore, by training our second-stage code learning by first running cross-validation on the first-stage predictors, we attempt to learn correcting codes which minimize the cross-validation classification error.

3.4.7 Platt's sigmoid method

Platt (1999) proposed a method for estimating posterior probabilities from SVM outputs in order to enable various kinds of post-processing. By converting the outputs of one-vs-the-rest SVM classifiers to class-specific posterior probabilities, in principle the probabilities are comparable and multi-class prediction through a one-vs-all strategy should improve. Platt's approach involves fitting a sigmoid for the posterior distribution,

$$P(y = 1|f) = \frac{1}{1 + \exp(Af + B)},$$

using training data of the form $\{(f_i, t_i)\}$, where f_i is the output of a trained SVM and t_i is the 0 or 1 target probability. The parameters A and B are found by maximizing the log likelihood of this training data. Typically, one would want to use a held-out set or cross-validation to generate outputs for fitting the sigmoid, since the outputs for the examples used to train the SVM give a biased estimate of the true output distribution. In addition to Platt's original sigmoid fitting algorithm, Lin et al. (2003) have proposed a more robust procedure.

However, in some cases, the sigmoid may be a poor description of the posterior probability, or there may be too little positive training data to properly fit the sigmoid. In our preliminary results (Ie et al., 2005), we found that sigmoid fitting performed poorly in our problem setting on a smaller data set. We retest here on larger benchmarks and again find that sigmoid fitting does not improve over one-vs-all for this problem (see Section 4).

3.4.8 In defense of one-vs-all

A recent empirical study suggests that the simple one-vs-all approach performs as well or better than all other multi-class methods in most cases (Rifkin and Klautau, 2004) when all the methods are well-tuned. However, the authors use only data sets with relatively few classes (between 4 and 48) for comparison and readily admit that that they use "toy data sets" from the UCI repository. Intuitively, the one-vs-all approach can be quite brittle in the many-class case: if only a single classifier is "corrupted" and always outputs a high score, then all of the examples can be misclassified. The more classes one has, the more chance that such corruptions can take place. In multi-class protein prediction, one has hundreds or thousands of classes. We present experimental results to show that the adaptive code approach improves over the "one-vs-all" by reweighting and effectively correcting such mistakes (see Section 4). Moreover, our approach also offers control of the loss function (such as using balanced loss) and use of hierarchical labels, which are not possible in one-vs-all.

4. Experimental Results

4.1 Data sets

We assembled benchmark data sets for the remote homology detection and fold recognition problems using sequences from the SCOP 1.65 protein database (see Section 2.1 for a definition of these problems). We used ASTRAL (Brenner et al., 2000) to filter these sequences so that no two sequences share greater than 95% identity.

For the fold recognition problem, we designed our experiments so that the test set consists of held-out superfamilies belonging to folds that are represented in the training data. We prepared a data set by first removing all superfamilies that have less than 5 sequence examples. We then removed all folds that have less than 3 superfamilies. We selected superfamilies for testing at random from the remaining superfamilies such that the test set for the superfamily contains no more than 40% of the remaining sequences for the fold. If at least one suitable superfamily could not be found, then the fold was removed from the experiment. The resulting fold detection data set contains of 26 folds, 303 superfamilies, and 652 families for training. We completely hold out 614 sequences from 46 superfamilies for testing.

For the remote homology detection, the test set should contain held-out families belonging to superfamilies that are represented in the training data. One can evaluate performance for multi-class prediction of fold or superfamily levels, and it is natural to try different codes for these two tasks; therefore, we prepared a separate data set for remote homology superfamily and fold detection. For the superfamily data set, we used the same selection scheme as for fold recognition, except the minimum number of sequences for the children of the superfamilies is relaxed to 3, and we selected random families for testing instead of superfamilies. The resulting superfamily detection data set contains of 74 superfamilies, and 544 families for training. We completely hold out 802 sequences from 110 families for testing.

For the remote homology fold detection data set, we first removed all superfamilies with less than 2 families. We then selected families from the remaining superfamilies for testing. We selected families at random from each superfamily such that we never selected more than 40% of the parent superfamily for testing. If no such families were found then the superfamily was removed from the data set. If a fold was then found to have no superfamilies with held out families for testing, it was removed from the data set. The resulting remote homology detection set contains 44 folds, 424 superfamilies, and 809 families for training. We completely hold out 381 sequences from 136 families for testing.

We use the training sequences in a cross-validation set-up to obtain classification scores and learn code weights. When training base classifiers, we only use negative data from outside of the target class of the experiment. For fold recognition, this means that when we train superfamily or family detectors, we exclude negative example sequences that come from the parent fold. We then retrain the base classifiers on all the training data to generate prediction scores for the test sequences, and then use the weighted code vectors to obtain multi-class predictions.

4.2 Methods

We test our weight learning approach using the ranking perceptron with the class-based, friend/foe, and mean class update rules for a variety of code sets for the remote homology detection and fold recognition problems. For each choice of codes, we compare against standard one-vs-all, sigmoid fitting using the robust procedure described in Lin et al. (2003), and a ranking perceptron version of the Crammer and Singer code learning method (Crammer and Singer, 2000). We do not test the SVM-struct implementation of our code learning optimization problem, since our preliminary results showed little difference in performance between the perceptron and SVM-struct on this problem (Ie et al., 2005).

As an additional baseline method, we also test PSI-BLAST, a widely used pairwise sequence comparison algorithm. In order to produce multi-class predictions, we use PSI-BLAST E-values as a distance measure for a nearest neighbor approach. PSI-BLAST E-values are not symmetric, since PSI-BLAST obtains somewhat different results depending on whether it builds a profile around each training sequence or each test sequence; however, preliminary results suggested that nearest neighbor performance was not significantly affected by this choice (Ie et al., 2005). Therefore, we use PSI-BLAST E-values based on training sequence profiles, which is the more computationally efficient choice.

For all ranking perceptron experiments, we train the perceptron algorithm for 200 iterations. When using SVM one-vs-all classifier codes, the learning parameter for all ranking

Method (and optimization target)	Error	Balanced Error	Top 5 Error	Balanced Top 5 Error
PSI-BLAST	0.399	0.457	0.273	0.365
one-vs-all: Sfams	0.271	0.445	0.105	0.197
one-vs-all: Sfams,Fams	0.271	0.445	0.110	0.207
Sigmoid Fitting: Sfams	0.365	0.547	0.197	0.369
Adaptive Codes: Sfams (zero-one)	0.247	0.385	0.096	0.148
Adaptive Codes: Sfams (balanced)	0.247	0.362	0.110	0.161
Adaptive Codes: Sfams,Fams (zero-one)	0.243	0.382	**0.090**	**0.141**
Adaptive Codes: Sfams,Fams (balanced)	0.239	0.352	0.107	0.162
Adaptive Codes: Sfams,Fams,PSI-Fams (zero-one)	0.223	0.338	0.094	0.142
Adaptive Codes: Sfams,Fams,PSI-Fams (balanced)	**0.217**	**0.320**	0.103	0.153

Table 1: **Results for multi-class superfamily prediction in the remote homology detection set-up.** Results for the adaptive code method are reported for a SCOP benchmark data set (67 folds, 74 superfamilies, 544 families, with 802 test sequences) and compared to nearest neighbor using PSI-BLAST, standard one-vs-all, and a perceptron version of the Crammer and Singer method. The mean class update rule is used to train the adaptive weights method.

perceptron experiments is set to 0.01, and the required margin is chosen to be $m = 2$. For ranking perceptron on one-vs-all classifier codes with PSI-BLAST extension, we set the initial weights on the PSI-BLAST portion of the codes to 0.1. We also use two learning parameters, 0.01 for the SVM portion and 0.001 for the PSI-BLAST portion. This choice effectively stops our algorithm from adjusting weights in the PSI-BLAST part of the code. We take this approach because the individual PSI-BLAST codes are derived from E-values and hence should already be comparable to each other. We use the same required margin of $m = 2$ in the ranking perceptron algorithm.

4.3 Remote homology detection results

For the remote homology detection data set, where the test set consists of held-out protein families that belong to superfamilies represented in the training data, we evaluate performance both for the superfamily-level and fold-level prediction tasks. Results for multi-class superfamily and fold prediction are provided in Tables 1 and 2, respectively. Significance tests are given comparing the methods in Tables 6, 7, 9, and 10. The last two tables use a balanced error measure by averaging the error rates over each prediction label before computing the significance test.

We compare our adaptive code method to PSI-BLAST, a standard homology detection method based on sequence alignment, as well as simple one-vs-all, sigmoid fitting, and the Crammer-Singer method, using various choices of code vectors. In addition to reporting classification loss and balanced loss results, we give "top 5" classification and balanced loss performance, which evaluates whether the correct class was found in the top 5 class predictions. The motivation for top 5 loss results is that a structural biologist might be willing to investigate a small number of false positives if it was likely that the list also contained the true structural class.

Method (and optimization target)	Error	Balanced Error	Top 5 Error	Balanced Top 5 Error
PSI-BLAST	0.409	0.443	0.297	0.367
one-vs-all: Folds	0.331	0.456	0.126	0.195
one-vs-all: Folds,Sfams	0.331	0.456	0.126	0.195
Sigmoid Fitting: Folds	0.339	0.514	0.163	0.329
Adaptive Codes: Folds (zero-one)	0.307	0.383	0.121	0.177
Adaptive Codes: Folds (balanced)	0.336	0.378	0.165	0.186
Adaptive Codes: Folds,Sfams (zero-one)	0.276	0.370	0.118	0.182
Adaptive Codes: Folds,Sfams (balanced)	0.297	0.351	0.134	0.173
Adaptive Codes: Folds,Sfams,Fams (zero-one)	**0.252**	0.351	**0.100**	0.168
Adaptive Codes: Folds,Sfams,Fams (balanced)	0.265	**0.340**	0.115	**0.142**

Table 2: **Results for multi-class fold prediction in the remote homology detection set-up.** Results for the adaptive codes method are reported for a SCOP benchmark data set (44 folds, 424 superfamilies, 809 families, with 381 test sequences) and compared to nearest neighbor using PSI-BLAST, standard one-vs-all, and a perceptron version of the Crammer and Singer method. The mean class update rule is used to train the adaptive weights method.

For the superfamily prediction task, we find that the adaptive codes method significantly outperforms one-vs-all both in terms of classification and balanced error, even when superfamily-only codes are used, and performance improves as more elements are added to the codes. By contrast, the Crammer-Singer code-learning method does not beat simple one-vs-all for this task, and performance tends to degrade as more elements are added to the codes. We also note that sigmoid fitting gives substantially worse performance than one-vs-all for this task. When compared to the widely-used PSI-BLAST method, even simple one-vs-all outperforms PSI-BLAST strongly in terms of classification error and slightly in terms of balanced error; adaptive codes outperforms PSI-BLAST very strongly by both measures and also when considering "top 5" prediction performance.

For the fold prediction task, we use a different set of codes, including code elements corresponding to protein fold detectors. We observe a similar trend, but with better results for Crammer-Singer when compared to one-vs-all. In this case, both Crammer-Singer and adaptive codes beat one-vs-all with respect to classification and balanced loss when fold-only codes are used; in fact, for fold-only codes, performance of Crammer-Singer is slightly better than adaptive codes. However, as we add more code elements, the performance of Crammer-Singer degrades while adaptive codes continues to improve, so that the best result for our method (corresponding to the longest code that we tried) is better than the best result for Crammer-Singer (the shortest code). The best results for both methods are significantly better than PSI-BLAST. Finally, sigmoid fitting slightly degrades performance as compared to one-vs-all.

Overall, we observe that when the individual code elements are helpful, as seems to be the case in remote homology detection, our adaptive code method can successfully improve performance by adding elements without overfitting. By constrast, the Crammer-Singer method, which learns a matrix of weights from the discriminant vectors to the label vectors, can perform well when codes are short but is susceptible to overfitting as they grow.

Method (and optimization target)	Error	Balanced Error	Top 5 Error	Balanced Top 5 Error
PSI-BLAST	0.648	0.703	0.518	0.543
one-vs-all: Folds	0.463	0.628	0.145	0.235
one-vs-all: Folds,Sfams	0.463	0.628	0.145	0.235
Sigmoid Fitting: Folds	0.451	0.628	0.169	0.287
Adaptive Codes: Folds (zero-one)	0.406	0.558	0.107	0.156
Adaptive Codes: Folds (balanced)	0.371	0.512	0.112	**0.145**
Adaptive Codes: Folds,Sfams (zero-one)	0.409	0.552	0.117	0.172
Adaptive Codes: Folds,Sfams (balanced)	**0.357**	0.508	0.109	0.146
Adaptive Codes: Folds,Sfams,Fams (zero-one)	0.401	0.535	**0.106**	0.173
Adaptive Codes: Folds,Sfams,Fams (balanced)	0.370	**0.499**	0.114	0.155

Table 3: **Results for multi-class fold prediction in the fold recognition set-up.** Results for the adaptive codes method are reported on a SCOP benchmark data set (26 folds, 303 superfamilies, 614 test sequences) and compared to nearest neighbor using PSI-BLAST, standard one-vs-all, and a perceptron version of the Crammer and Singer method. The adaptive codes method was trained using the mean class update rule.

4.4 Fold recognition results

For the more difficult fold recognition task, where the data set consists of held-out superfamilies from protein folds represented in the training data, we expect that code elements from subclasses (i.e. superfamilies and families) will provide less information, since protein sequences from different superfamilies in principle have no detectable sequence similarity.

Results for the fold recognition problem are provided in Table 3. Note first that the errors for PSI-BLAST, even for the top 5 fold predictions, are very high, underscoring the difficulty of the problem. Sigmoid fitting appears to slightly help reduce one-vs-all error in this case, though balanced error is unaffected. We find that the adaptive codes method can again beat one-vs-all and strongly outperform PSI-BLAST, but we see no trend of improvement as more code elements are added, with various length codes leading to similar error rates. The best classification error rate for adaptive codes is somewhat lower than the best one for Crammer-Singer. Interestingly, in this case, Crammer-Singer with fold-only codes outperforms the best adaptive codes result in terms of balanced loss, though the top 5 results for adaptive codes are uniformly better than Crammer-Singer by either loss function. We conclude that in this case, since the code elements corresponding to subclasses are not very helpful, the adaptive code method cannot leverage longer codes to achieve much higher accuracy. However, the weight learning approach does significantly outperform one-vs-all by all evaluation measures. Significance tests are given comparing the methods in Tables 8 and 11.

4.5 Modified perceptron update rules

Finally, for all multi-class prediction class, we evaluate the effectiveness of our modified perceptron update rules: the friend/foe rule and the mean class update rule. Results are shown in Table 4. Significance tests are given comparing the methods in Table 5.

	Error			Balanced Error		
Method (and optimization target)	single codes	friend/foe	mean friend/foe	single codes	friend/foe	mean friend/foe
Fold recognition						
Folds,Sfams (zero-one)	0.402	0.414	0.409	0.547	0.558	0.552
Folds,Sfams (balanced)	0.412	0.368	0.357	0.535	0.509	0.508
Folds,Sfams,Fams (zero-one)	0.404	0.406	0.401	0.548	0.553	0.535
Folds,Sfams,Fams (balanced)	0.406	0.378	0.370	0.497	0.508	0.499
Remote Homology Superfamily Prediction						
Sfams,Fams (zero-one)	0.251	0.239	0.243	0.372	0.380	0.382
Sfams,Fams (balanced)	0.266	0.241	0.239	0.385	0.347	0.352
Sfams,Fams,PSI-Fams (zero-one)	0.232	0.219	0.223	0.330	0.340	0.338
Sfams,Fams,PSI-Fams (balanced)	0.241	0.213	0.217	0.346	0.313	0.320
Remote Homology Fold Prediction						
Folds,Sfams (zero-one)	0.310	0.283	0.276	0.391	0.372	0.370
Folds,Sfams (balanced)	0.375	0.312	0.297	0.401	0.360	0.351
Folds,Sfams,Fams (zero-one)	0.252	0.247	0.252	0.363	0.347	0.351
Folds,Sfams,Fams (balanced)	0.315	0.278	0.265	0.373	0.349	0.340

Table 4: **Results for multi-class prediction comparing different perceptron update rules.** Results for the friend/foe and mean friend/foe update rules are compared with the standard perceptron update rule for the fold recognition and remote homology fold and superfamily prediction tasks when using hierarchical codes. Experiments shown with a gray background are those for which the modified update rule gives poorer performance than the standard rule, usually by an insignificant amount. In all other experiments, the modified rules consistently outperform the regular rule, usually by a significant amount (but with one case of a tie).

	Superfamily			Fold			Fold (Remote Homology)		
	single Codes	friend/foe	mean friend/foe	single Codes	friend/foe	mean friend/foe	single Codes	friend/foe	mean friend/foe
single Codes	1	-	-	1	0.73	0.92	1	-	-
friend/foe	0.03	1	0.85	-	1	-	0.01	1	-
mean friend/foe	0.08	-	1	-	0.22	1	0	0.12	1

Table 5: **P-values from the Wilcoxon signed rank test for different perceptron update rules.** The table shows, at the 0.05 significance level, whether a method in a given row beats a method in a given column (numbers with gray background are significant). Dashes represent when a method in a given row did not beat the method in the given column. All measurments are for balanced error.

	PSI-BLAST	one-vs-all: Folds	one-vs-all: Folds,Sfams	Sigmoid Fitting: Folds	C&S: Sf (balanced)	C&S: Sf,f (balanced)	C&S: Sf,f,PSI-f (balanced)	A-Codes: Sf (balanced)	A-Codes: Sf,f (balanced)	A-Codes: Sf,f,PSI-f (balanced)
PSI-BLAST	1	-	-	-	-	-	0.61	-	-	-
one-vs-all: Folds	0	1	1	0	0	0	0	-	-	-
one-vs-all: Folds,Sfams	0	1	1	0	0	0	0	-	-	-
Sigmoid Fitting: Folds	0.06	-	-	1	-	0.25	0	-	-	-
C&S: Sf (balanced)	0	-	-	0	1	0	0	-	-	-
C&S: Sf,f (balanced)	0.25	-	-	-	-	1	0.05	-	-	-
C&S: Sf,f,PSI-f (balanced)	-	-	-	-	-	-	1	-	-	-
A-Codes: Sf (balanced)	0	0.01	0.01	0	0	0	0	1	-	-
A-Codes: Sf,f (balanced)	0	0	0	0	0	0	0	0.13	1	-
A-Codes: Sf,f,PSI-f (balanced)	0	0	0	0	0	0	0	0	0	1

Table 6: **P-values from the Wilcoxon signed rank test for superfamily prediction in the remote homology setup.** The table shows, at the 0.05 significance level, whether a method in a given row beats a method in a given column (numbers with gray background are significant). Dashes represent when a method in a given row did not beat the method in the given column.

	PSI-BLAST	one-vs-all: Folds	one-vs-all: Folds,Sfams	Sigmoid Fitting: Folds	C&S: F (balanced)	C&S: F,Sf (balanced)	C&S: F,Sf,f (balanced)	A-Codes: F (balanced)	A-Codes: F,Sf (balanced)	A-Codes: F,Sf,f (balanced)
PSI-BLAST	1	-	-	-	-	0	0	-	-	-
one-vs-all: Folds	0	1	1	0.59	0.05	0	0	0.77	-	-
one-vs-all: Folds,Sfams	0	1	1	0.59	0.05	0	0	0.77	-	-
Sigmoid Fitting: Folds	0.01	-	-	1	0.1	0	0	-	-	-
C&S: F (balanced)	0.24	-	-	-	1	0	0	-	-	-
C&S: F,Sf (balanced)	-	-	-	-	-	1	-	-	-	-
C&S: F,Sf,f (balanced)	-	-	-	-	-	0	1	-	-	-
A-Codes: F (balanced)	0.01	-	-	0.89	0.03	0	0	1	-	-
A-Codes: F,Sf (balanced)	0	0.05	0.05	0.04	0	0	0	0	1	-
A-Codes: F,Sf,f (balanced)	0	0	0	0	0	0	0	0	0	1

Table 7: **P-values from the Wilcoxon signed rank test for fold prediction in the remote homology setup.** The table shows, at the 0.05 significance level, whether a method in a given row beats a method in a given column (numbers with gray background are significant). Dashes represent when a method in a given row did not beat the method in the given column.

	PSI-BLAST	one-vs-all: Folds	one-vs-all: Folds,Sfams	Sigmoid Fitting: Folds	C&S: F (balanced)	C&S: F,Sf (balanced)	C&S: F,Sf,f (balanced)	A-Codes: F (balanced)	A-Codes: F,Sf (balanced)	A-Codes: F,Sf,f (balanced)
PSI-BLAST	1	-	-	-	-	-	-	-	-	-
one-vs-all: Folds	0	1	1	-	-	0.07	0	-	-	-
one-vs-all: Folds,Sfams	0	1	1	-	-	0.07	0	-	-	-
Sigmoid Fitting: Folds	0	0.39	0.39	1	-	0.02	0	-	-	-
C&S: F (balanced)	0	0	0	0	1	0	0	-	-	-
C&S: F,Sf (balanced)	0	-	-	-	-	1	0	-	-	-
C&S: F,Sf,f (balanced)	0	-	-	-	-	-	1	-	-	-
A-Codes: F (balanced)	0	0	0	0	0.56	0	0	1	-	-
A-Codes: F,Sf (balanced)	0	0	0	0	0.15	0	0	0.03	1	0.21
A-Codes: F,Sf,f (balanced)	0	0	0	0	0.48	0	0	0.88	-	1

Table 8: **P-values from the Wilcoxon signed rank test for fold recognition.** The table shows, at the 0.05 significance level, whether a method in a given row beats a method in a given column (numbers with gray background are significant.) Dashes represent when a method in a given row did not beat the method in the given column.

	PSI-BLAST	one-vs-all: Folds	one-vs-all: Folds,Sfams	Sigmoid Fitting: Folds	C&S: Sf (balanced)	C&S: Sf,f (balanced)	C&S: Sf,f,PSI-f (balanced)	A-Codes: Sf (balanced)	A-Codes: Sf,f (balanced)	A-Codes: Sf,f,PSI-f (balanced)
PSI-BLAST	1	-	-	0.03	-	0.07	0.08	-	-	-
one-vs-all: Folds	0.79	1	1	0	0.4	0.01	0.02	-	-	-
one-vs-all: Folds,Sfams	0.79	1	1	0	0.4	0.01	0.02	-	-	-
Sigmoid Fitting: Folds	-	-	-	1	-	-	-	-	-	-
C&S: Sf (balanced)	0.86	-	-	0	1	0.01	0.07	-	-	-
C&S: Sf,f (balanced)	-	-	-	0.55	-	1	-	-	-	-
C&S: Sf,f,PSI-f (balanced)	-	-	-	0.35	-	0.44	1	-	-	-
A-Codes: Sf (balanced)	0.01	0	0	0	0	0	0	1	-	-
A-Codes: Sf,f (balanced)	0.01	0	0	0	0	0	0	0.09	1	-
A-Codes: Sf,f,PSI-f (balanced)	0	0	0	0	0	0	0	0.02	0.05	1

Table 9: **P-values from the Wilcoxon signed rank test for balanced superfamily prediction in the remote homology setup.** The table shows, at the 0.05 significance level, whether a method in a given row beats a method in a given column (numbers with gray background are significant). Dashes represent when a method in a given row did not beat the method in the given column.

	PSI-BLAST	one-vs-all: Folds	one-vs-all: Folds,Sfams	Sigmoid Fitting: Folds	C&S: F (balanced)	C&S: F,Sf (balanced)	C&S: F,Sf,f (balanced)	A-Codes: F (balanced)	A-Codes: F,Sf (balanced)	A-Codes: F,Sf,f (balanced)
PSI-BLAST	1	0.95	0.95	0.34	-	0.34	0.73	-	-	-
one-vs-all: Folds	-	1	1	0.39	-	0.34	-	-	-	-
one-vs-all: Folds,Sfams	-	1	1	0.39	-	0.34	-	-	-	-
Sigmoid Fitting: Folds	-	-	-	1	-	-	-	-	-	-
C&S: F (balanced)	0.04	0.05	0.05	0.01	1	0.01	0.04	0.84	0.67	-
C&S: F,Sf (balanced)	-	-	-	1	-	1	-	-	-	-
C&S: F,Sf,f (balanced)	-	0.78	0.78	0.38	-	0.39	1	-	-	-
A-Codes: F (balanced)	0.12	0.01	0.01	0.01	-	0.02	0.13	1	-	-
A-Codes: F,Sf (balanced)	0.02	0	0	0	-	0.01	0.03	0.03	1	-
A-Codes: F,Sf,f (balanced)	0.01	0	0	0	0.43	0.01	0.02	0.01	0.09	1

Table 10: **P-values from the Wilcoxon signed rank test for balanced fold prediction in the remote homology setup.** The table shows, at the 0.05 significance level, whether a method in a given row beats a method in a given column (numbers with gray background are significant). Dashes represent when a method in a given row did not beat the method in the given column.

	PSI-BLAST	one-vs-all: Folds	one-vs-all: Folds,Sfams	Sigmoid Fitting: Folds	C&S: F (balanced)	C&S: F,Sf (balanced)	C&S: F,Sf,f (balanced)	A-Codes: F (balanced)	A-Codes: F,Sf (balanced)	A-Codes: F,Sf,f (balanced)
PSI-BLAST	1	-	-	-	-	-	-	-	-	-
one-vs-all: Folds	0.16	1	1	0.92	-	-	-	-	-	-
one-vs-all: Folds,Sfams	0.16	1	1	0.92	-	-	-	-	-	-
Sigmoid Fitting: Folds	0.36	-	-	1	-	-	-	-	-	-
C&S: F (balanced)	0	0.01	0.01	0.01	1	0.03	0	0.12	0.19	0.21
C&S: F,Sf (balanced)	0.16	0.58	0.58	0.49	-	1	0.24	-	-	-
C&S: F,Sf,f (balanced)	0.7	0.66	0.66	0.96	-	-	1	-	-	-
A-Codes: F (balanced)	0	0	0	0.01	-	0.21	0.02	1	-	-
A-Codes: F,Sf (balanced)	0	0	0	0.01	-	0.17	0.03	0.81	1	-
A-Codes: F,Sf,f (balanced)	0	0	0	0.01	-	0.12	0.03	0.73	0.52	1

Table 11: **P-values from the Wilcoxon signed rank test for balanced fold recognition.** The table shows, at the 0.05 significance level, whether a method in a given row beats a method in a given column (numbers with gray background are significant). Dashes represent when a method in a given row did not beat the method in the given column.

We find that the new update rules consistently and significantly improve performance for both remote homology prediction tasks when evaluated in terms of classification error, with the most dramatic improvements occurring when training the perceptron using balanced loss in the remote homology fold prediction task. The same performance improvement is true when measured in terms of balanced error for the remote homology fold prediction task; however, for remote homology superfamily prediction, the improvement in balanced error only holds when the perceptron is also trained with balanced error.

In the case of fold recognition, previous results indicate that the subclass code elements are less useful, so we expect that update rules which respect the code structure may be less effective. Indeed, we get mixed results here, with a neutral or slightly weakening effect when the perceptrons are trained using classification loss. However, even for fold recognition, the new update rules significantly improve classification error when the perceptrons are trained using balanced loss.

5. Discussion

We have presented a novel and effective method for multi-class classification that uses the ranking perceptron to learn a reweighting of components of output vectors. Our application domain is the highly multi-class protein structural classification problem, where there are typically hundreds of classes arranged in a hierarchical taxonomy. In this domain, we focus on two difficult subproblems: remote homology detection and fold recognition. We exploit hierarchical information in this problem by training one-vs-the-rest SVM classifiers to recognize classes at different levels of the hierarchy and using these classifiers to define different components of the output vectors. We then use fixed binary codes to represent the hierarchy of labels associated with each class, and we adapt our output vector embedding in order to improve classification relative to these fixed codes.

Unlike the results of a recent empirical study of multi-class classification algorithms that used smaller "toy data sets" (Rifkin and Klautau, 2004), we find that we can significantly outperform one-vs-all in our problem setting. We also convincingly beat PSI-BLAST, which is a widely-used alignment-based method for detecting relationships between protein sequences.

Many authors have presented "output code" methods for multi-class classification. We compare our approach to a perceptron version of the recent Crammer-Singer code-learning approach, which seeks to learn a mapping from the vector of prediction scores for an input example to the vector of output classes. We find that when there are a smaller number of classes and when relatively few code elements are used, the Crammer-Singer method can tie or slightly outperform (but not statistically significantly) our adaptive code approach. However, as the number of code elements grows, Crammer-Singer performance deteriorates, often giving much poorer results than one-vs all, while our performance continues to improve. Therefore, as we add more base classifiers, we can almost always beat the best Crammer-Singer result. Moreover, we also present results using modified update rules for the ranking perceptron which take into consideration multi-class predictions that lead to the same loss. These update rules, called the friend/foe and mean friend/foe updates, lead to small but significant performance advantages across multiple experiments.

A large body of recent work has focused on finding good representations for the *inputs* in the protein classification problem, in particular in the form of novel string kernels for protein sequence data. Our current study focuses on the complementary problem of adapting the embedding of the *outputs*. Our experimental results provide a promising indication that new kernel methods combined with novel multi-class "output space" algorithms can truly achieve state-of-the-art performance in a large-scale multi-class protein classification setting.

As we scale to the full-scale protein classification problem, with on the order of 1000 folds, one issue with our approach is limited coverage: for many small SCOP folds, there are not enough labeled sequences to train an SVM fold detector. In ongoing work, we are considering two strategies for increasing coverage. First, there is a standard method for increasing the positive training set size in this problem, namely using PSI-BLAST or another alignment-based method to pull in additional sequences from the non-redundant database that are homologous to the known fold members. Adding domain homologs creates a larger, if biased, training set, and one could investigate the trade-off between coverage and multi-class accuracy as one applies this strategy to very small classes. Second, we are investigating a strategy of "punting" from one prediction method to another based on a prediction score threshold. The goal is to combine a method with weaker performance but full coverage, such as PSI-BLAST, with a higher accuracy method with reduced coverage, such as SVM adaptive codes, to produce a hybrid method with full coverage that in general outperforms both component methods. Details and results of this approach will be reported elsewhere.

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 and NIH grant GM74257 and by the NIH Roadmap Initiative, National Centers for Biomedical Computing Grant 1U54CA121852.

References

Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, pages 9–16. Morgan Kaufmann, San Francisco, CA, 2000.

Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

Zafer Barutcuoglu, Robert E. Schapire, and Olga G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.

Asa Ben-Hur and Douglas Brutlag. Remote homology detection: a motif based approach. *Proceedings of the Eleventh International Conference on Intelligent Systems for Molecular Biology*, 19 suppl 1:i26–i33, 2003.

Léon Bottou, Yann LeCun, and Yoshua Bengio. Global training of document processing systems using graph transformer networks. In *Proc. of Computer Vision and Pattern Recognition*, pages 490–494, Puerto-Rico, 1997. IEEE.

Steven E. Brenner, Patrice Koehl, and Michael Levitt. The ASTRAL compendium for sequence and structure analysis. *Nucleic Acids Research*, 28:254–256, 2000.

Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research*, 7:31–54, 2006.

Michael Collins. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning*, pages 175 – 182. Morgan Kaufmann, San Francisco, CA, 2000.

Michael Collins and Nigel Duffy. 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*, pages 263–270, 2002.

Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. In *Computational Learning Theory*, pages 35–46, 2000.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal Machine Learning Research*, 2:265–292, 2002. ISSN 1533-7928.

Ofer Dekel, Joseph Keshet, and Yoram Singer. Large margin hierarchical classification. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.

Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277 – 296, 1999.

Mark Girolami and Simon Rogers. Variational bayesian multinomial probit regression with gaussian process priors. *Neural Computation*, 18(8):1790–1817, 2006.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

Eugene Ie, Jason Weston, William Stafford Noble, and Christina Leslie. Multi-class protein fold recognition using adaptive codes. *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

Tommi Jaakkola, Mark Diekhans, and David Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1–2):95–114, 2000.

Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjölander, and David Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

Rui Kuang, Eugene Ie, Ke Wang, Kai Wang, Mahira Siddiqi, Yoav Freund, and Christina Leslie. Profile kernels for detecting remote protein homologs and discriminative motifs. *Journal of Bioinformatics and Computational Biology*, 2005. To appear.

Yann LeCun and Fu Jie Huang. Loss functions for discriminative training of energy-based models. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, 2005.

Christina Leslie, Eleazar Eskin, and William S. Noble. The spectrum kernel: A string kernel for SVM protein classification. *Proceedings of the Pacific Biocomputing Symposium*, pages 564–575, 2002a.

Christina Leslie, Eleazar Eskin, Jason Weston, and William S. Noble. Mismatch string kernels for SVM protein classification. *Advances in Neural Information Processing Systems 15*, pages 1441–1448, 2002b.

Christina Leslie, Eleazar Eskin, Adiel Cohen, Jason Weston, and William S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004.

Li Liao and William S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. *Proceedings of the 6th Annual International Conference on Research in Computational Molecular Biology*, pages 225–232, 2002.

Hsuan-Tien Lin, Chih-Jen Lin, and Ruby C. Weng. A note on platt's probabilistic outputs for support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, 2003.

Alexey G. Murzin, Steven E. Brenner, Tim Hubbard, and Cyrus Chothia. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536–540, 1995.

Jong Park, Kevin Karplus, Christian Barrett, Richard Hughey, David Haussler, Tim Hubbard, and Cyrus Chothia. Sequence comparisons using multiple sequences detect twice as many remote homologues as pairwise methods. *Journal of Molecular Biology*, 284(4): 1201–1210, 1998.

John Platt. Probabilities for support vector machines. *Advances in Large Margin Classifiers*, pages 61–74, 1999.

Huzefa Rangwala and George Karypis. Profile-based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, 21(23):4239–4247, 2005.

Gunnar Rätsch, Alexander J. Smola, and Sebastian Mika. Adapting codes and embeddings for polychotomies. *Advances in Neural Information Processing Systems*, 15:513–520, 2002.

Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal Machine Learning Research*, 5:101–141, 2004. ISSN 1533-7928.

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

Hiroto Saigo, Jean-Philippe Vert, Nobuhisa Ueda, and Tatsuya Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004.

Temple Smith and Michael Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector learning for interdependent and structured output spaces. *Proceedings of the 21st International Conference on Machine Learning*, pages 823–830, 2004.

Vladimir N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, 1998.

Jason Weston and Chris Watkins. Support vector machines for multiclass pattern recognition. In *Proceedings of the 7th European Symposium On Artificial Neural Networks*, 1999.

Jason Weston, Christina Leslie, Eugene Ie, Dengyong Zhou, Andre Elisseeff, and William S. Noble. Semi-supervised protein classification using cluster kernels. *Bioinformatics*, 21 (15):3241–3247, 2005.