Predicting Structured Objects with Support Vector Machines

Thorsten Joachims
Dept. of Computer Science
Cornell University
Ithaca, NY 14853, USA
tj@cs.cornell.edu

Thomas Hofmann
Google Inc.
Brandschenkestrasse 110
8002 Zürich, Switzerland
thofmann@google.com

Yisong Yue
Dept. of Computer Science
Cornell University
Ithaca, NY 14853, USA
yyue@cs.cornell.edu

Chun-Nam Yu
Dept. of Computer Science
Cornell University
Ithaca, NY 14853, USA
cnyu@cs.cornell.edu

ABSTRACT

Machine Learning today offers a broad repertoire of methods for classification and regression. But what if we need to predict complex objects like trees, orderings, or alignments? Such problems arise naturally in natural language processing, search engines, and bioinformatics. The following explores a generalization of Support Vector Machines (SVMs) for such complex prediction problems.

1. INTRODUCTION

Consider the problem of natural language parsing illustrated in Figure 1 (left). A parser takes as input a natural language sentence, and the desired output is the parse tree decomposing the sentence into its constituents. While modern machine learning methods like Boosting, Bagging, and Support Vector Machines (SVMs) (see e.g. [9]) have become the methods of choice for other problems in natural language processing (NLP) (e.g. word-sense disambiguation), parsing does not fit into the conventional framework of classification and regression. In parsing, the prediction is not a single yes/no, but a labeled tree. And breaking the tree down into a collection of yes/no predictions is far from straightforward, since it would require modeling a host of inter-dependencies between individual predictions. So, how can we take, say, an SVM and learn a rule for predicting trees?

Obviously, this question arises not only for learning to predict trees, but similarly for a variety of other structured and complex outputs. *Structured output prediction* is the name for such learning tasks, where one aims at learning a function $h : \mathcal{X} \rightarrow \mathcal{Y}$ mapping inputs $\mathbf{x} \in \mathcal{X}$ to complex and structured outputs $\mathbf{y} \in \mathcal{Y}$. In NLP, structured output prediction tasks range from predicting an equivalence relation in noun-phrase co-reference resolution (see Figure 1, right), to predicting an accurate and well-formed translation in machine translation. But problems of this type are also plentiful beyond NLP. For example, consider the problem of image segmentation (i.e. predicting an equivalence relation $\mathbf{y}$ over a matrix of pixels $\mathbf{x}$), the problem of protein structure prediction (which we will phrase as an alignment problem in Section 3.2), or the problem of web search (i.e. predicting a diversified document ranking $\mathbf{y}$ given a query $\mathbf{x}$ as explored in Section 3.1). We will see in Section 3.3 that even binary classification becomes a structured prediction task when aiming to optimize multivariate performance measures like the F_1-Score or $Precision@k$.

In this paper we describe a generalization of SVMs, called *Structural SVMs* [26, 27, 14], that can be used to address a large range of structured output prediction tasks. On the one hand, structural SVMs inherit the attractive properties of regular SVMs, namely a convex training problem, flexibility in the choice of loss function, and the opportunity to learn non-linear rules via kernels. On the other hand, structural SVMs inherit the expressive power of generative models (e.g. Probabilistic Context-Free Grammars (PCFG) or Markov Random Fields (MRF)). Most importantly, however, structural SVMs are a discriminative learning method that does not require the independence assumptions made by conventional generative methods. Similar to the increase in prediction accuracy one typically sees when switching from a naive Bayes classifier to a classification SVM (e.g. [11]), structural SVMs promise such benefits also for structured prediction tasks (e.g. training PCFGs for parsing).

Structural SVMs follow and build upon a line of research on discriminative training for structured output prediction, including generalizations of Neural Nets [17], Logistic Regression [16], and other conditional likelihood methods (e.g. [18]). A particularly eye-opening paper is [16], showing that global training for structured prediction can be formulated as a convex optimization problem. Our work follows this track. More closely, however, we build upon structural Perceptrons [5] as well as methods for protein threading [19] and extend these to a large-margin formulation with an efficient training algorithm. Independent of our work, Taskar et al. [25] arrived at a similar formulation, but with more restrictive conditions on the form of the learning task.

In the following, we explain how structural SVMs work and highlight three applications in search engine ranking, protein structure prediction, and binary classification under non-standard performance measures.

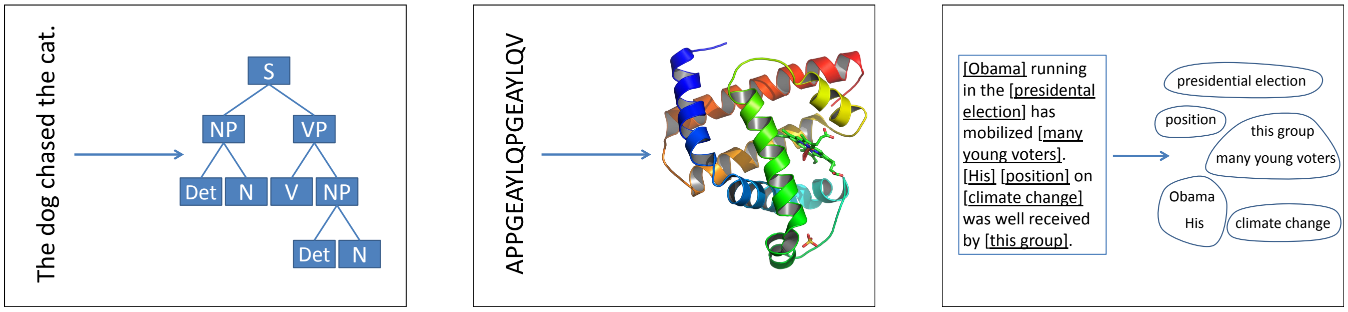
The dog chased the cat.
S
NP
VP
Det
N
V
NP
Det
N
APPGEAYLQPGEAYLQV
[Obama] running in the [presidental election] has mobilized [many young voters]. [His] [position] on [climate change] was well received by [this group].
presidential election
position
this group
many young voters
Obama
His
climate change

Figure 1: Examples of structured output prediction tasks: predicting trees in natural language parsing (left), predicting the structure of proteins (middle), and predicting an equivalence relation over noun phrases (right).

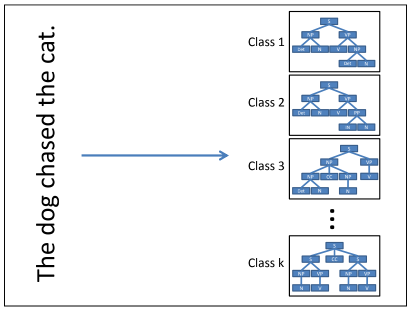

Figure 2: Structured output prediction as a multi-class problem.

2. STRUCTURAL SVMS

How can one approach structured output prediction? On an abstract level, a structured prediction task is much like a multi-class learning task. Each possible structure $\mathbf{y} \in \mathcal{Y}$ (e.g. parse tree) corresponds to one class (see Figure 2), and classifying a new example $\mathbf{x}$ amounts to predicting its correct "class". While the following derivation of structural SVMs starts from multi-class SVMs [6], there are four key problems that need to be overcome. All of these problems arise from the huge number $|\mathcal{Y}|$ of classes. In parsing, for example, the number of possible parse trees is exponential in the length of the sentence. And the situation is similar for most other structured output prediction problems.

The first problem is related to finding a compact representation for large output spaces. If we allowed even just one parameter for each class, we would already have more parameters than we could ever hope to have enough training data for. Second, just making a single prediction on a new example is a computationally challenging problem, since sequentially enumerating all possible classes may be infeasible. Third, we need a more refined notion of what constitutes a prediction error. Clearly, predicting a parse tree that is almost correct should be treated differently from predicting a tree that is completely wrong. And, last but not least, we need efficient training algorithms that have a run-time complexity sub-linear in the number of classes.

In the following, we will tackle these problems one by one, starting with the formulation of the structural SVM method.

2.1 Problem 1: Structural SVM Formulation

As mentioned above, we start the derivation of the structural SVM from the multi-class SVM [6]. These multi-class SVMs use one weight vector $\boldsymbol{w}_{\mathbf{y}}$ for each class $\mathbf{y}$. Each input $\mathbf{x}$ now has a score for each class $\mathbf{y}$ via $f(\mathbf{x},\mathbf{y}) \equiv \boldsymbol{w}_{\mathbf{y}} \cdot \Phi(\mathbf{x})$. Here $\Phi(\mathbf{x})$ is a vector of binary or numeric features extracted from $\mathbf{x}$. Thus, every feature will have an additively weighted influence in the modeled compatibility between inputs $\mathbf{x}$ and classes $\mathbf{y}$. To classify $\mathbf{x}$, the prediction rule $h(\mathbf{x})$ then simply chooses the highest-scoring class

$$h(\mathbf{x}) \equiv \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} f(\mathbf{x},\mathbf{y}) \qquad (1)$$

as the predicted output. This will result in the correct prediction $\mathbf{y}$ for input $\mathbf{x}$ provided the weights $\boldsymbol{w} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k)$ have been chosen such that the inequalities $f(\mathbf{x},\bar{\mathbf{y}}) < f(\mathbf{x},\mathbf{y})$ hold for all incorrect outputs $\bar{\mathbf{y}} \neq \mathbf{y}$.

For a given training sample $(\mathbf{x}_1,\mathbf{y}_1)$, ..., $(\mathbf{x}_n,\mathbf{y}_n)$, this leads directly to a (hard-)margin formulation of the learning problem by requiring a fixed margin (=1) separation of all training examples, while using the norm of $\boldsymbol{w}$ as a regularizer:

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2, \text{ s.t. } f(\mathbf{x}_i,\mathbf{y}_i) - f(\mathbf{x}_i,\bar{\mathbf{y}}) \geq 1 \quad (\forall i, \bar{\mathbf{y}} \neq \mathbf{y}_i) \quad (2)$$

For a k-class problem, the optimization problem has a total of $n(k-1)$ inequalities that are all linear in $\boldsymbol{w}$, since one can expand $f(\mathbf{x}_i,\mathbf{y}_i) - f(\mathbf{x}_i,\bar{\mathbf{y}}) = (\boldsymbol{w}_{\mathbf{y}_i} - \boldsymbol{w}_{\bar{\mathbf{y}}}) \cdot \Phi(\mathbf{x}_i)$. Hence, it is a convex quadratic program.

The first challenge in using (2) for structured outputs is that, while there is generalization across inputs $\mathbf{x}$, there is *no generalization across outputs*. This is due to having a separate weight vector $\boldsymbol{w}_{\mathbf{y}}$ for each class $\mathbf{y}$. Furthermore, since the number of possible outputs can become very large (or infinite), naively reducing structured output prediction to multi-class classification leads to an undesirable blow-up in the overall number of parameters.

The key idea in overcoming these problems is to extract features from input-output pairs using a so-called *joint feature map* $\Psi(\mathbf{x},\mathbf{y})$ instead of $\Phi(\mathbf{x})$. This yields compatibility functions with contributions from combined properties of inputs and outputs. These joint features will allow us to generalize across outputs and to define meaningful scores even for outputs that were never actually observed in the training data. At the same time, since we will define compatibility functions via $f(\mathbf{x},\mathbf{y}) \equiv \boldsymbol{w} \cdot \Psi(\mathbf{x},\mathbf{y})$, the number of parameters will simply equal the number of features extracted via Ψ, which may not depend on $|\mathcal{Y}|$. One can then use the formulation in (2) with the more flexible definition of f via Ψ to arrive at the following (hard-margin) optimization problem for structural SVMs.

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \tfrac{1}{2}\|\boldsymbol{w}\|^2 \qquad (3) \\ \text{s.t.} \quad & \boldsymbol{w}\cdot\Psi(\mathbf{x}_i,\mathbf{y}_i) - \boldsymbol{w}\cdot\Psi(\mathbf{x}_i,\bar{\mathbf{y}}) \geq 1 \quad (\forall i, \bar{\mathbf{y}} \neq \mathbf{y}_i) \end{aligned}$$

In words, find a weight vector $\boldsymbol{w}$ of an input-ouput compatibility function f that is linear in some joint feature map Ψ so that on each training example it scores the correct output higher by a fixed margin than every alternative output, while having low complexity (i.e. small norm $\|\boldsymbol{w}\|$). Note that the number of linear constraints is still $n(|\mathcal{Y}|-1)$, but we will suggest ways to cope with this efficiently in Section 2.4.

The design of the features Ψ is problem-specific, and it is a strength of the developed methods to allow for a great deal of flexibility in how to choose it. In the parsing example above, the features extracted via Ψ may consist of counts of how many times each production rule of the underlying grammar has been applied in the derivation described by a pair $(\mathbf{x},\mathbf{y})$. For other applications, the features can be derived from graphical models as proposed in [16, 25], but more general features can be used as well.

2.2 Problem 2: Efficient Prediction

Before even starting to address the efficiency of solving a quadratic programs of the form (3) for large n and $|\mathcal{Y}|$, we need to make sure that we can actually solve the inference problem of computing a prediction $h(\mathbf{x})$. Since the number of possible outputs $|\mathcal{Y}|$ may grow exponentially in the length of the representation of outputs, a brute-force exhaustive search over $\mathcal{Y}$ may not always be feasible. In general, we require some sort of structural matching between the (given) compositional structure of the outputs $\mathbf{y}$ and the (designed) joint feature map Ψ.

For instance, if we can decompose $\mathcal{Y}$ into non-overlapping independent parts, $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_m$ and if the feature extraction Ψ respects this decomposition such that no feature combines properties across parts, then we can maximize the compatibility for each part separately and compose the full output by simple concatenation. A significantly more general case can be captured within the framework of Markov networks as explored by [16, 25]. If we represent outputs as a vector of (random) variables $\mathbf{y} = (y_1,\ldots,y_m)$, then for a fixed input $\mathbf{x}$, we can think of $\Psi(\mathbf{x},\mathbf{y})$ as sufficient statistics in a conditional exponential model of $P(\mathbf{y}|\mathbf{x})$. Maximizing $f(\mathbf{x},\mathbf{y})$ then corresponds to finding the most probable output, $\hat{\mathbf{y}} = \mathrm{argmax}_{\mathbf{y}}\, P(\mathbf{y}|\mathbf{x})$. Modeling the statistical dependencies between output variables vis a dependency graph, one can use efficient inference methods such as the junction tree algorithm for prediction. This assumes the clique structure of the dependency graph induced by a particular choice of Ψ is tractable (e.g. the state space of the largest cliques remains sufficiently small). Other efficient algorithms, for instance, based on dynamic programming or minimal graph cuts, may be suitable for other prediction problems.

In our example of natural language parsing, the suggested feature map will lead to a compatibility function in which parse trees are scored by associating a weight with each production rule and by simply combining all weights additively. Therefore, a prediction can be computed efficiently using the CKY-Parsing algorithm (cf. [27]).

2.3 Problem 3: Inconsistent Training Data

So far we have tacitly assumed that the optimization problem in (3) has a solution, i.e. there exists a weight vector that simultaneously fulfills all margin constraints. In practice this may not be the case, either because the training data is inconsistent or because our model class is not powerful enough. If we allow for mistakes, though, we must be able to quantify the degree of mismatch between a prediction and the correct output, since usually different incorrect predictions vary in quality. This is exactly the role played by a *loss function*, formally $\Delta : \mathcal{Y}\times\mathcal{Y} \to \Re$, where $\Delta(\mathbf{y},\bar{\mathbf{y}})$ is the loss (or cost) for predicting $\bar{\mathbf{y}}$, when the correct output is $\mathbf{y}$. Ideally we are interested in minimizing the expected loss of a structured output classifier, yet, as is common in machine learning, one often uses the empirical or training loss $\frac{1}{n}\sum_{i=1}^{n}\Delta(\mathbf{y}_i, h(\mathbf{x}_i))$ as a proxy for the (inaccessible) expected loss. Like the choice of Ψ, defining Δ is problem-specific. If we predict a set or sequence of labels, a natural choice for Δ may be the number of incorrect labels (a.k.a. Hamming loss). In the above parsing problem, a quality measure like F_1 may be the preferred way to quantify the similarity between two labeled trees in terms of common constituents (and then one may set $\Delta \equiv 1 - F_1$).

Now we will utilize the idea of loss functions to refine our initial problem formulation. Several approaches have been suggested in the literature, all of which are variations of the so-called soft-margin SVM: instead of strictly enforcing constraints, one allows for violations, yet penalizes them in the overall objective function. A convenient approach is to introduce slack variables that capture the actual violation,

$$f(\mathbf{x}_i,\mathbf{y}_i) - f(\mathbf{x}_i,\bar{\mathbf{y}}) \geq 1 - \xi_{i\bar{\mathbf{y}}}\,, (\forall \bar{\mathbf{y}} \neq \mathbf{y}_i) \qquad (4)$$

so that by choosing $\xi_{i\bar{\mathbf{y}}} > 0$ a smaller (even negative) separation margin is possible. All that remains to be done then is to define a penalty for the margin violations. A popular choice is to use $\frac{1}{n}\sum_{i=1}^{n}\max_{\bar{\mathbf{y}}\neq\mathbf{y}_i}\{\Delta(\mathbf{y}_i,\bar{\mathbf{y}})\xi_{i\bar{\mathbf{y}}}\}$, thereby penalizing violations more heavily for constraints associated with outputs $\bar{\mathbf{y}}$ that have a high loss with regard to the observed $\mathbf{y}_i$. Technically, one can convert this back into a quadratic program as follows:

$$\min_{\boldsymbol{w},\xi_i\geq 0} \; \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \qquad (5)$$
$$\text{s.t.} \;\; \boldsymbol{w}\cdot\Psi(\mathbf{x}_i,\mathbf{y}_i) - \boldsymbol{w}\cdot\Psi(\mathbf{x}_i,\bar{\mathbf{y}}) \geq 1 - \frac{\xi_i}{\Delta(\mathbf{y}_i,\bar{\mathbf{y}})}\,, (\forall i, \bar{\mathbf{y}} \neq \mathbf{y}_i)$$

Here $C > 0$ is a constant that trades-off constraint violations with the geometric margin (effectively given by $1/\|\boldsymbol{w}\|$). This has been called the slack re-scaling formulation. As an alternative, one can also re-scale the margin to upper-bound the training loss as first suggested in [25] and discussed in [27]. This leads to the following quadratic program

$$\min_{\boldsymbol{w},\xi_i\geq 0} \; \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \qquad (6)$$
$$\text{s.t.} \;\; \boldsymbol{w}\cdot\Psi(\mathbf{x}_i,\mathbf{y}_i) - \boldsymbol{w}\cdot\Psi(\mathbf{x}_i,\bar{\mathbf{y}}) \geq \Delta(\mathbf{y}_i,\bar{\mathbf{y}}) - \xi_i\,, (\forall i, \bar{\mathbf{y}} \neq \mathbf{y}_i)$$

Since it is slightly simpler, we will focus on this margin-rescaling formulation in the following.

2.4 Problem 4: Efficient Training

Last, but not least, we need a training algorithm that finds the optimal $\boldsymbol{w}$ solving the quadratic program in (6). Since there is a contraint for every incorrect label $\bar{\mathbf{y}}$, we cannot enumerate all constraints and simply give the optimization

Algorithm 1 for training structural SVMs (margin-rescaling).

```
1: Input: $S = ((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))$, $C$, $\epsilon$
2: $\mathcal{W} \leftarrow \emptyset$, $\boldsymbol{w} = \mathbf{0}$, $\xi_i \leftarrow 0$ for all $i = 1, ..., n$
3: repeat
4:   for i=1,...,n do
5:     $\hat{\mathbf{y}} \leftarrow \text{argmax}_{\hat{\mathbf{y}} \in \mathcal{Y}}\{\Delta(\mathbf{y}_i, \hat{\mathbf{y}}) + \boldsymbol{w} \cdot \Psi(\mathbf{x}_i, \hat{\mathbf{y}})\}$
6:     if $\boldsymbol{w} \cdot [\Psi(\mathbf{x}_i, \mathbf{y}_i) - \Psi(\mathbf{x}_i, \hat{\mathbf{y}})] < \Delta(\mathbf{y}_i, \hat{\mathbf{y}}) - \xi_i - \epsilon$ then
7:       $\mathcal{W} \leftarrow \mathcal{W} \cup \{\boldsymbol{w} \cdot [\Psi(\mathbf{x}_i, \mathbf{y}_i) - \Psi(\mathbf{x}_i, \hat{\mathbf{y}})] \geq \Delta(\mathbf{y}_i, \hat{\mathbf{y}}) - \xi_i\}$
8:       $(\boldsymbol{w}, \boldsymbol{\xi}) \leftarrow \text{argmin}_{\boldsymbol{w}, \boldsymbol{\xi} \geq \mathbf{0}} \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} + \frac{C}{n}\sum_{i=1}^{n} \xi_i$ s.t. $\mathcal{W}$
9:     end if
10:  end for
11: until $\mathcal{W}$ has not changed during iteration
12: return($\boldsymbol{w}$,$\boldsymbol{\xi}$)
```

problem to a standard QP solver. Instead, we propose to use the cutting-plane Algorithm 1 (or a similar algorithm for slack-rescaling). The key idea is to iteratively construct a working set of constraints $\mathcal{W}$ that is equivalent to the full set of constraints in (6) up to a specified precision ϵ. Starting with an empty $\mathcal{W}$ and $\boldsymbol{w} = 0$, Algorithm 1 iterates through the training examples. For each example, the argmax in Line 5 finds the most violated constraint of the quadratic program in (6). If this constraint is violated by more than ϵ (Line 6), it is added to the working set $\mathcal{W}$ in Line 7 and a new $\boldsymbol{w}$ is computed by solving the quadratic program over the new $\mathcal{W}$ (Line 8). The algorithm stops and returns the current $\boldsymbol{w}$ if $\mathcal{W}$ did not change between iterations.

It is obvious that for any desired ϵ, the algorithm only terminates when it has found an ϵ-accurate solution, since it verifies in Line 5 that none of the constraints of the quadratic program in (6) is violated by more than ϵ. But how long does it take to terminate? It can be shown [27] that Algorithm 1 always terminates in a polynomial number of iterations that is independent of the cardinality of the output space $\mathcal{Y}$. In fact, a refined version of Algorithm 1 [13, 14] always terminates after adding at most $O(C\epsilon^{-1})$ constraints to $\mathcal{W}$ (typically $|\mathcal{W}| << 1000$). Note that the number of constraints is not only independent of $|\mathcal{Y}|$, but also independent of the number of training examples n, which makes it an attractive training algorithm even for conventional SVMs [13].

While the number of iterations is small, the argmax in Line 5 might be expensive to compute. In general, this is true, but note that this argmax is closely related to the argmax for computing a prediction $h(\mathbf{x})$. It is therefore called the "loss-augmented" inference problem, and often the prediction algorithm can be adapted to efficiently solve the loss-augmented inference problem as well.

Note that Algorithm 1 is rather generic and refers to the output structure only through a narrow interface. In particular, to apply the algorithm to a new structured prediction problem, one merely needs to supply implementations of $\Psi(\mathbf{x}, \mathbf{y})$, $\Delta(\mathbf{y}, \bar{\mathbf{y}})$, and $\text{argmax}_{\bar{\mathbf{y}} \in \mathcal{Y}}\{\Delta(\mathbf{y}_i, \bar{\mathbf{y}}) + \boldsymbol{w} \cdot \Psi(\mathbf{x}_i, \bar{\mathbf{y}})\}$. This makes the algorithm general and easily transferable to new applications. In particular, all applications discussed in the following used the general SVM^{struct} implementation of Algorithm 1 and supplied these three functions via an API.

2.5 Related Approaches

The structural SVM method is closely related to other approaches, namely Conditional Random Fields [16] and Maximum Margin Markov networks (M3N) [25]. The crucial link to probabilistic methods is to interpret the joint feature map as sufficient statistics in a conditional exponential family model. This means, we define a conditional probability of outputs given inputs by

$$P(\mathbf{y}|\mathbf{x}) = \tfrac{1}{Z(\mathbf{x})} \exp\left[\boldsymbol{w} \cdot \Psi(\mathbf{x}, \mathbf{y})\right] , \qquad (7)$$

where $Z(\mathbf{x})$ is a normalization constant. Basically (7) is a generalization of the well-known logistic regression where $\Psi(\mathbf{x}, \mathbf{y}) = \mathbf{y}\Phi(\mathbf{x})$ for $\mathbf{y} \in \{-1, 1\}$. The compatibility function used in structural SVMs directly governs this conditional probability. The probabilistic semantics lends itself to statistical inference techniques like penalized (conditional) maximum likelihood, where one maximizes $\sum_{i=1}^{n} \log P(\mathbf{y}_i|\mathbf{x}_i) - \lambda\|\boldsymbol{w}\|^2$. Unlike in Algorithm 1, however, here one usually must compute the expected sufficient statistics $\sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})\Psi(\mathbf{x}, \mathbf{y})$, for instance, in order to compute a gradient direction on the conditional log-likelihood. There are additional algorithmic challenges involved with learning CRFs (cf. [22]) and one of the key advantages of structural SVMs is that it allows an efficient dual formulation. This enables the use of kernels instead of explicit feature representations as well as a sparse expansion of the solution (cf. [10]). A simpler, but usually less accurate learning algorithm that can be generalized to the structured output setting is the perceptron algorithm, as first suggested in [5].

The M3N approach is also taking the probabilistic view as its starting point, but instead of maximizing a likelihood-based criterion, aims at solving (6). More specifically, M3Ns exploit the clique-based representation of $P(\mathbf{y}|\mathbf{x})$ over a dependency graph to efficiently re-parametrize the dual problem of (6), effectively replacing the margin constraints over pairs of outputs by constraints involving functions of clique configurations. This is an interesting alternative to our cutting-plane algorithm, but has a somewhat more limited range of applicability.

3. APPLICATIONS

As outlined above, one needs to design and supply the following four functions when applying a structural SVM to a new problem:

$$\Psi(\mathbf{x}, \mathbf{y}) \qquad (8)$$

$$\Delta(\mathbf{y}, \bar{\mathbf{y}}) \qquad (9)$$

$$\text{argmax}_{\bar{\mathbf{y}} \in \mathcal{Y}}\{\boldsymbol{w} \cdot \Psi(\mathbf{x}_i, \bar{\mathbf{y}})\} \qquad (10)$$

$$\text{argmax}_{\bar{\mathbf{y}} \in \mathcal{Y}}\{\Delta(\mathbf{y}_i, \bar{\mathbf{y}}) + \boldsymbol{w} \cdot \Psi(\mathbf{x}_i, \bar{\mathbf{y}})\} \qquad (11)$$

The following three sections illustrate how this can be done for learning retrieval functions that provide diversified results, for aligning amino-acid sequences into homologuous protein structures, and for optimizing non-standard performance measures in binary classification.

3.1 Optimizing Diversity in Search Engines

State of the art retrieval systems commonly use machine learning to optimize their ranking functions. While effective to some extent, conventional learning methods score each document independently and, therefore, cannot account for information diversity (e.g. not presenting multiple redundant results). Indeed, several recent studies in information retrieval emphasized the need for diversity (e.g., [32, 24]). In particular, they stressed modeling inter-document dependencies, which is fundamentally a structured prediction

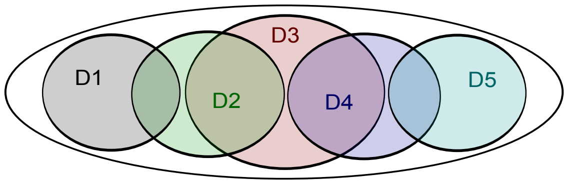

Figure 3: Different documents (shaded regions) cover different information (subtopics) of a query. Here, the set $\{D2, D3, D4\}$ contains the 3 documents that individually cover the most information, but $\{D1, D3, D5\}$ collectively covers more information.

problem. Given a dataset of queries and documents labeled according to information diversity, we used structural SVMs to learn a general model of how to diversify [31].

What is an example $(\mathbf{x}, \mathbf{y})$ in this learning problem? For some query, let $\mathbf{x} = \{x_1, \ldots, x_n\}$ be the candidate documents that the system needs to rank. Our ground truth labeling for $\mathbf{x}$ is a set of subtopics $\mathbf{T} = \{T_1, \ldots, T_n\}$, where T_i denotes the subtopics covered by document x_i. The prediction goal is to find a subset $\mathbf{y} \subset \mathbf{x}$ of size K (e.g., $K = 10$ for search) maximizing subtopic coverage, and therefore maximizing the information presented to the user. We define our loss function $\Delta(\mathbf{T}, \mathbf{y})$ to be the (weighted) fraction of subtopics not covered by $\mathbf{y}$ (more weight for popular subtopics)[1].

Even if the ground truth subtopics were known, computing the best $\mathbf{y}$ can be difficult. Since documents overlap in subtopics (i.e., $\exists i, j : T_i \cap T_j \neq \emptyset$), this is a budgeted maximum coverage problem. The standard greedy method achieves a $(1-1/e)$-approximation bound [15], and typically yields good solutions. The greedy method also naturally produces a ranking of the top documents.

Figure 3 depicts an abstract visualization of our prediction problem. The shaded regions represent candidate documents $\mathbf{x}$ of a query, and the area covered by each region is the "information" (represented as subtopics $\mathbf{T}$) covered by that document. If $\mathbf{T}$ was known, we could use a greedy method to find a solution with high subtopic diversity. For $K = 3$, the optimal solution in Figure 3 is $\mathbf{y} = \{D1, D3, D5\}$.

In general, however, the subtopics of new queries are unknown. One can think of subtopic labels as a manual partitioning of the total information of a query. We do however, have access to an "automatic" representation of information diversity: the words contained in the documents. Intuitively, covering more (distinct) words should result in covering more subtopics. Since some words are more informative than others, a weighting scheme is required.

Following the approach of Essential Pages [24], we measure word importance primarily using relative word frequencies within $\mathbf{x}$. For example, the word "computer" is generally informative, but conveys almost no information for the query "ACM" since it likely appears in most of the candidate documents. We learn a word weighting function using labeled training data, whereas [24] uses a pre-defined function.

We now present a simple example of the joint feature representation Ψ. Let $\phi(v, \mathbf{x})$ denote the feature vector describing the frequency of a word v amongst documents in $\mathbf{x}$. For

[1]The loss function (9) can be defined using any ground truth format, not necessarily the same format as the output space.

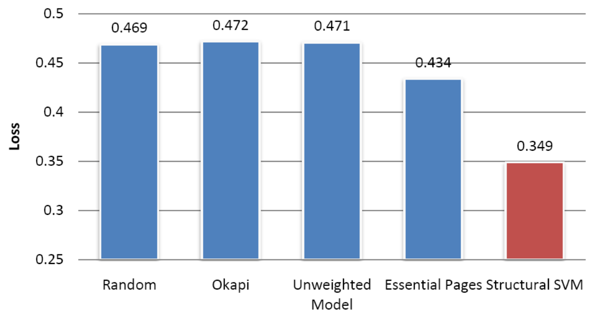

Figure 4: Subtopic loss comparison when retrieving 5 documents. Structural SVM performance is superior with 95% confidence (using signed rank test).

example, we can construct $\phi(v, \mathbf{x})$ as

$$\phi(v, \mathbf{x}) = \begin{pmatrix} \mathbf{1}[v \text{ appears in at least 15\% of } \mathbf{x}] \\ \mathbf{1}[v \text{ appears in at least 35\% of } \mathbf{x}] \\ \ldots \end{pmatrix}.$$

Let $V(\mathbf{y})$ denote the union of words contained in the documents of the predicted subset $\mathbf{y}$. We can write Ψ as

$$\Psi(\mathbf{x}, \mathbf{y}) = \sum_{v \in V(\mathbf{y})} \phi(v, \mathbf{x}).$$

Given a model vector $\boldsymbol{w}$, the benefit of covering word v in $\mathbf{x}$ is $\boldsymbol{w} \cdot \phi(v, \mathbf{x})$, and is realized when a document in $\mathbf{y}$ contains v, i.e., $v \in V(\mathbf{y})$. Because documents overlap in words, this is also budgeted maximum coverage problem. Thus both prediction (10) and loss-augmented inference (11) can be solved effectively via the greedy method. Despite finding an approximate most violated constraint, we can still bound the precision of our solution [8]. Practical applications require more sophisticated Ψ and we refer to [31] for more details.

Experiments. We tested the effectiveness of our method using the TREC 6-8 Interactive Track queries. Relevant documents are labeled using subtopics. For example, query 392 asked human judges to identify applications of robotics in the world today, and they identified 36 subtopics among the results such as nanorobots and using robots for space missions. Additional details can be found in [31].

We compared against Okapi [21], Essential Pages [24], random selection and unweighted word selection (all words have equal benefit). Okapi is a conventional retrieval function which does not account for diversity. Figure 4 shows the loss on the test set for retrieving $K = 5$ documents. The gains of the structural SVM over Essential Pages are 95% significant, and only the structural SVM and Essential Pages outperformed random.

This approach can accomodate other diversity criteria, such as diversity of clicks gathered from clickthrough logs. We can also accommodate very rich feature spaces based on, e.g., anchor text, URLs, and titles.

Abstractly, we are learning a mapping between two representations of the same set covering problem. One representation defines solution quality using manually labeled subtopics. The second representation learns a word weighting function, with goal of having the representations agree on the best solution. This setting is very general and can be applied to other domains beyond subtopic retrieval.

3.2 Predicting Protein Alignments

Proteins are sequences of 20 different amino acids joined together by peptide bonds. They fold into various stable structures under normal cell environments. A central question in biology is to understand how proteins fold, as their structures relate to their functions.

One of the more successful methods for predicting protein structure from an amino acid sequence is homology modeling. It approximates the structure of an unknown protein by comparing it against a database of proteins with experimentally determined structures. An important intermediate step is to align the unknown protein sequence with known protein sequences in the database, and this is a difficult problem when the sequences have diverged too far (e.g., less than 20% sequence identity).

To align protein sequences accurately for homology modeling, we need to have a good substitution cost matrix as input to the Smith-Waterman algorithm (a dynamic programming algorithm). A common approach is to learn the substitution matrix from a set of known good alignments between evolutionarily related proteins.

Structural SVMs are particularly suitable for learning the substitution matrix, since they allow incorporating all available information (e.g. secondary structures, solvent accessibility, and other physiochemical properties) in a principled way. When predicting the alignment $\mathbf{y} = (y_1, y_2, ...)$ for two given protein sequences $\mathbf{x} = (s_a, s_b)$, each sequence location is described by a vector of features. The discriminative approach of structural SVMs makes it easy to incorporate these extra features without having to make unjustified independence assumptions as in generative models. As explained below, this enables learning a 'richer' substitution function $\boldsymbol{w} \cdot \Phi(\mathbf{x}, y_i)$ instead of a fixed substitution matrix.

The Smith-Waterman algorithm uses a linear function for scoring alignments, which allows us to decompose the joint feature vector into a sum of feature vectors for individual alignment operations (match, insertion, or deletion):

$$\boldsymbol{w} \cdot \Psi(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{\text{length}(\mathbf{y})} \boldsymbol{w} \cdot \Phi(\mathbf{x}, y_i)$$

$\Phi(\mathbf{x}, y_i)$ is the feature vector for the ith alignment operation in the alignment $\mathbf{y}$ of the sequence pair $\mathbf{x}$. Below we describe several alignment models represented by Φ, focusing on the scoring of matching two amino acids (see Figure 5 for an illustration of the features used):

(i) *Simple*: we only consider the substitution cost of single features, e.g., the cost of matching a position with amino acid 'M' with another position in a loop region.

(ii) *Anova2*: we consider pairwise interaction of features, e.g., the cost of matching a position with amino acid 'M' in an alpha helix with another position with amino acid 'V' in a loop region.

(iii) *Tensor*: we consider three-way interaction of features among amino acid, secondary structure, and solvent accessibility.

(iv) *Window*: on top of three alignment models above, we add neighborhood features using a sliding window, which takes into account the hydrophobicity and secondary structure in the neighborhood of a position.

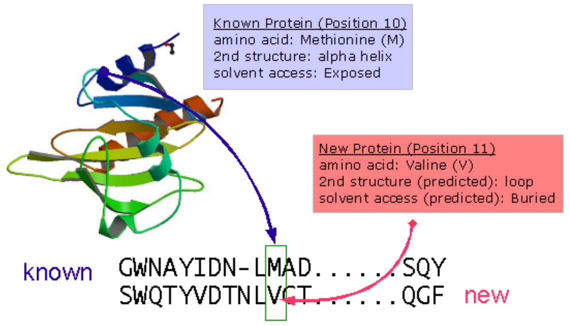
Known Protein (Position 10)
amino acid: Methionine (M)
2nd structure: alpha helix
solvent access: Exposed
New Protein (Position 11)
amino acid: Valine (V)
2nd structure (predicted): loop
solvent access (predicted): Buried
known
GWNAYIDN-LMAD.......SQY
SWQTYVDTNLVCT.......QGF
new

Figure 5: Aligning a new protein sequence with a known protein structure. Features at the aligned positions are used to construct substitution matrices under different alignment models.

(v) *Profile*: on top of the alignment model *Window*, we add PSI-BLAST profile scores as features.

As the loss function $\Delta(\mathbf{y}, \bar{\mathbf{y}})$, we use Q_4-loss. It is the percentage of matched amino acids in the correct alignment $\mathbf{y}$ that are aligned more than 4 positions apart in the predicted alignment $\bar{\mathbf{y}}$. The linearity of the Q_4-loss allows us to use the Smith-Waterman algorithm also for solving the loss-augmented inference problem during training. We refer to [29] for more details.

Experiments. We tested our algorithm with the training and validation sets developed in [20], which contain 4542 and 4587 pairwise alignments of evolutionarily-related proteins with high structural similarites. For the test set we selected 4185 structures deposited in the Protein Data Bank from June 2005 to June 2006, leading to 29345 test pairs. For all sequence pairs $\mathbf{x}$ in the training, validation, and test sets both structures are known, and we use the CE structural alignment program [23] to generate "correct" alignments $\mathbf{y}$.

The red bars in Figure 6 shows the Q_4-scores (i.e. 100 minus Q_4-loss) of the five alignment models described above. By carefully introducing features and considering their interactions, we can build highly complex alignment models (with hundreds of thousands of parameters) with very good alignment performance. Note that a conventional substitution matrix has only $20^2 = 400$ parameters.

SSALN[20] (blue bar) uses a generative alignment model for parameter estimation, and is trained using the same training set and features as the structural SVM algorithm. The structural SVM model *Window* substantially outperforms SSALN on Q_4-score. Incorporating profile information makes the SVM model *Profile* perform even better, illustrating the benefit of being able to easily add additional features in discriminative training. The result from BLAST (green bar) is provided as a baseline.

To get a plausible upper bound for further improvements, we check in how far the CE alignments we used as ground truth agree with the structural alignments computed by TM-Align [33]. TM-Align gives a Q_4-score of 85.45 when using the CE alignments as ground truth to compare against. This provides a reasonable upper bound on the alignment accuracy we might achieve on this noisy dataset. However, it also shows significant room for further research and improvement from our current alignment models.

3.3 Binary Classification with Unbalanced Classes

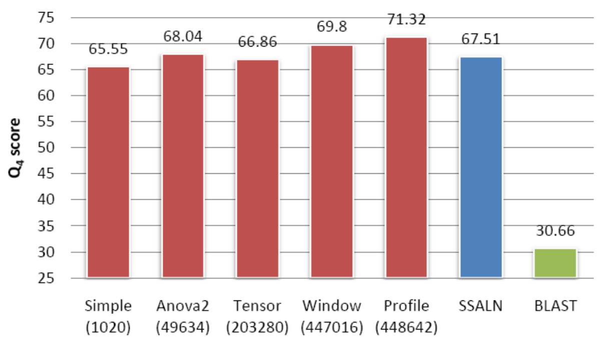

Figure 6: Q_4-score of various Structural SVM alignment models compared to two baseline models. The Structural SVM using Window or Profile features significantly outperforms the SSALN baseline. The number in brackets is the number of features of the corresponding alignment model.

Even binary classification problems can become structured output problems in some cases. Consider, for example, the case of learning a binary text classification rule with the classes "machine learning" and "other topics". Like most text classification task it is highly unbalanced, and we might only have, say, 1% machine-learning documents in our collection. In such a situation, prediction error typically becomes meaningless as a performance measure, since the default classifier that always predicts "other topics" already has a great error rate that is hard to beat. To overcome this problem, the Information Retrieval (IR) community has designed other performance measures, like the F_1-Score and the Precision/Recall Breakeven Point (PRBEP) (see e.g. [1]), that are meaningful even with unbalanced classes.

What does this mean for learning? Instead of optimizing some variant of error rate during training, which is what conventional SVMs and virtually all other learning algorithms do, it seems like a natural choice to have the learning algorithm directly optimize, for example, the F_1-Score (i.e. the harmonic mean of precision and recall). This is the point where our binary classification task needs to become a structured output problem, since F_1-Score (as well as many other IR measures) are not functions of individual examples (like error rate), but a function of a set of examples. In particular, we arrive at the structured output problem of predicting an array of labels $\mathbf{y} = (y_1, ..., y_n)$, $y_i \in \{-1, +1\}$ for an array of feature vectors $\mathbf{x} = (x_1, ..., x_n)$, $x_i \in \Re^N$. Each possible array of labels $\bar{\mathbf{y}}$ now has an associated F_1-Score $F_1(\mathbf{y}, \bar{\mathbf{y}})$ w.r.t. the true labeling $\mathbf{y}$, and optimizing F_1-Score on the training set becomes a well-defined problem.

The problem of predicting an array of binary labels $\mathbf{y}$ for an array of input vectors $\mathbf{x}$ optimizing a loss function $\Delta(\mathbf{y}, \bar{\mathbf{y}})$ fits neatly into the structural SVM framework. Again, we only need to define $\Psi(\mathbf{x}, \mathbf{y})$ and $\Delta(\mathbf{y}, \bar{\mathbf{y}})$ appropriately, and then find algorithms for the two argmax problems. The choice of loss function is obvious: when optimizing F_1-Score, which ranges from 0 (worst) to 1 (best), we will use

$$\Delta(\mathbf{y}, \bar{\mathbf{y}}) = 1 - F_1(\mathbf{y}, \bar{\mathbf{y}}).$$

Table 1: Comparing a conventional SVM to a structual SVM that directly optimizes the performance measure.

Dataset	Method	F_1	PRBEP	ROCArea
Reuters	struct SVM	62.0	68.2	99.1
	SVM	56.1	65.7	98.6
ArXiv	struct SVM	56.8	58.4	92.8
	SVM	49.6	57.9	92.7
Optdigits	struct SVM	92.5	92.7	99.4
	SVM	91.5	91.5	99.4
Covertype	struct SVM	73.8	72.1	94.6
	SVM	73.9	71.0	94.1

For the joint feature mapping, using

$$\Psi(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} y_i x_i$$

can be shown to be a natural choice, since it makes the structural SVM equivalent to a conventional SVM if error rate is used as the loss $\Delta(\mathbf{y}, \bar{\mathbf{y}})$. It also makes computing a prediction very efficient with

$$\hat{\mathbf{y}}(\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmax}} \{\boldsymbol{w} \cdot \Psi(\mathbf{x}, \mathbf{y})\} = (\operatorname{sign}(\boldsymbol{w} \cdot x_1), ..., \operatorname{sign}(\boldsymbol{w} \cdot x_n)).$$

Computing the loss-augmented argmax necessary for training is a bit more complicated and depends on the particular loss function used, but it can be done in time at most $O(n^2)$ for any loss function that can be computed from the contingency table of prediction errors [12]. SVM^{perf} is an implementation of the method for various performance measures.

Experiments. Table 1 shows the prediction performance of the structural SVM on four benchmark datasets, described in more detail in [12]. In particular, it compares the structural SVM that directly optimizes the measure it is evaluated on (here F_1, PRBEP, and ROCArea) to a conventional SVM that accounts for unbalanced classes with a linear cost model. For both methods, parameters were selected to optimize the evaluation measure via cross validation. In most cases, the structural SVM outperforms the conventional SVM, and it never does substantially worse.

Beyond these performance gains, the structural SVM approach has an attractive simplicity to it – direct optimization of the desired performance measure instead of using proxy measures during training (e.g. different linear cost models). However, there are still several open question before making this process fully automatic; for example, understanding the interaction between Δ and Ψ as recently explored in [4, 3].

4. CONCLUSIONS AND OUTLOOK

In summary, structural SVMs are flexible and efficient methods for structured output prediction with a wide range of potential applications, most of which are completely or largely unexplored. Other explored applications include hierarchical classification [2], clustering [7], and optimizing average precision [30]. Due to the universality of the cutting-plane training algorithm, only relatively small API changes are required for any new application. SVM^{struct} is an implementation of the algorithm with APIs in C and Python, and it is available at `svmlight.joachims.org`.

Beyond applications, there are several areas for research in further developing the method. One such area is training structural SVMs with Kernels. While the cutting-plane method can be extended to use Kernels, it becomes quite slow and more efficient methods are needed [28]. Another area results from that fact that solving the two argmax problems exactly is intractable for many application problems. However, for many such problems there exist methods that compute approximate solutions. An interesting question is how the approximation quality affects the quality of the structural SVM solution [8].

This work was supported in part through NSF Awards IIS-0412894 and NSF IIS-0713483, NIH Grants IS10RR020889 and GM67823, a gift from Yahoo!, and by Google.

5. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley-Longman, Harlow, UK, May 1999.

[2] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, 2004.

[3] S. Chakrabarti, R. Khanna, U. Sawant, and C. Battacharyya. Structured learning for non-smooth ranking losses. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2008.

[4] O. Chapelle, Q. Le, and A. Smola. Large margin optimization of ranking measures. In *NIPS Workshop on Machine Learning for Web Search*, 2007.

[5] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8, 2002.

[6] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research (JMLR)*, 2:265–292, 2001.

[7] T. Finley and T. Joachims. Supervised clustering with support vector machines. In *International Conference on Machine Learning (ICML)*, pages 217–224, 2005.

[8] T. Finley and T. Joachims. Training structural SVMs when exact inference is intractable. In *International Conference on Machine Learning (ICML)*, pages 304–311, 2008.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[10] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220, 2008.

[11] T. Joachims. *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms.* Kluwer/Springer, 2002.

[12] T. Joachims. A support vector method for multivariate performance measures. In *International Conference on Machine Learning (ICML)*, pages 377–384, 2005.

[13] T. Joachims. Training linear SVMs in linear time. In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, pages 217–226, 2006.

[14] T. Joachims, T. Finley, and C.-N. Yu. Cutting-plane training of structural svms. *Machine Learning*, to appear.

[15] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1997.

[16] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, 2001.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[18] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *International Conference on Machine Learning (ICML)*, pages 591–598, 2000.

[19] J. Meller and R. Elber. Linear programming optimization and a double statistical filter for protein threading protocols. *Proteins Structure, Function, and Genetics*, 45:241–261, 2001.

[20] J. Qiu and R. Elber. SSALN: an alignment algorithm using structure-dependent substitution matrices and gap penalties learned from structurally aligned protein pairs. *Proteins*, 62:881–91, 2006.

[21] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of TREC-3*, 1994.

[22] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 134–141, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[23] I. N. Shindyalov and P. E. Bourne. Protein structure alignment by incremental combinatorial extension(CE) of the optimal path. *Protein Eng*, 11:739–747, 1998.

[24] A. Swaminathan, C. Mathew, and D. Kirovski. Essential pages. Technical Report MSR-TR-2008-015, Microsoft Research, 2008.

[25] B. Taskar, C. Guestrin, and D. Koller. Maximum-margin markov networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[26] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning (ICML)*, pages 104–112, 2004.

[27] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)*, 6:1453–1484, September 2005.

[28] C.-N. J. Yu and T. Joachims. Training structural svms with kernels using sampled cuts. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 794–802, 2008.

[29] C.-N. J. Yu, T. Joachims, R. Elber, and J. Pillardy. Support vector training of protein alignment models. *Journal of Computational Biology*, 15(7):867–880,

September 2008.

[30] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 271–278, 2007.

[31] Y. Yue and T. Joachims. Predicting diverse subsets using structural SVMs. In *International Conference on Machine Learning (ICML)*, pages 271–278, 2008.

[32] C. Zhai, W. W. Cohen, and J. Lafferty. Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval. In *Proceedings of the ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 2003.

[33] Y. Zhang and J. Skolnick. TM-align: A protein structure alignment algorithm based on TM-score. *Nucleic Acids Research*, 33:2302–2309, 2005.