IMPROVED LEARNING OF STRUCTURAL SUPPORT VECTOR MACHINES: TRAINING WITH LATENT VARIABLES AND NONLINEAR KERNELS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Chun Nam Yu

January 2011

IMPROVED LEARNING OF STRUCTURAL SUPPORT VECTOR MACHINES: TRAINING WITH LATENT VARIABLES AND NONLINEAR KERNELS

Chun Nam Yu, Ph.D.

Cornell University 2011

Structured output prediction in machine learning is the study of learning to predict complex objects consisting of many correlated parts, such as sequences, trees, or matchings. The Structural Support Vector Machine (Structural SVM) algorithm is a discriminative method for structured output learning that allows flexible feature construction with robust control for overfitting. It provides state-of-art prediction accuracies for many structured output prediction tasks in natural language processing, computational biology, and information retrieval.

This thesis explores improving the learning of structured prediction rules with structural SVMs in two main areas: incorporating latent variables to extend their scope of application and speeding up the training of structural SVMs with nonlinear kernels. In particular, we propose a new formulation of structural SVM, called Latent Structural SVM, that allows the use of latent variables, and an algorithm to solve the associated non-convex optimization problem. We demonstrate the generality of our new algorithm through several structured output prediction problems, showing improved prediction accuracies with new alternative problem formulations using latent variables.

In addition to latent variables, the use of nonlinear kernels in structural SVMs can also improve their expressiveness and prediction accuracies. However their high computational costs during training limit their wider application. We explore the use of approximate cutting plane models to speed up

the training of structural SVMs with nonlinear kernels. We provide a theoretical analysis of their iteration complexity and their approximation quality. Experimental results show improved accuracy-sparsity tradeoff when compared against several state-of-art approximate algorithm for training kernel SVMs, with our algorithm having the advantage that it is readily applicable to structured output prediction problems.

BIOGRAPHICAL SKETCH

Chun-Nam Yu was born and grew up in Hong Kong, spending the first two decades of his life there. In 2001 he travelled to England to pursue university education, that being the very first time he stepped outside Asia. After overcoming the weather and the food he began to appreciate and enjoy studying inside the historic buildings at Oxford, and loved the weekly academic discussions with his tutors in Wadham College. He earned a BA degree in Mathematics and Computer Science with First Class Honours in 2004, and then moved across the Atlantic to pursue a PhD degree in Computer Science at Cornell University. There he saw real snow for the first time and studied under the guidance of Professor Thorsten Joachims. He learned to do research in machine learning, working mainly in the area of support vector machines and structured output learning. After finishing his PhD he continues his trajectory towards the northwest to brave colder weather and will become a postdoctoral fellow at the Alberta Ingenuity Center for Machine Learning at the University of Alberta in Canada.

To my family

ACKNOWLEDGEMENTS

First of all I must thank my PhD advisor Professor Thorsten Joachims, for his mentorship and friendship over these years. Looking back, his wise and patient guidance over these six years transformed me from an inexperienced undergraduate to an independent researcher ready to tackle problems on his own. It is hard to describe either my gratitude or my good luck in having Thorsten as my advisor. His taste and choice of problems, discipline and commitment to hard work, all had a great influence on me. His patience during my ups and downs, his encouragement to me to try new and bolder approaches and problems, all made my PhD journey much smoother and at the same time much more exciting.

I would also like to thank my committee members, Professor Adam Siepel and Professor Mike Todd. Their careful reading of this thesis helps improve its presentation greatly. I want to thank Adam for teaching me graphical models and many things in computational biology, and Mike for teaching me optimization and for his discussions when I encountered issues involving optimization in my research. Many other mentors also helped me greatly in my development during these PhD years, to whom I want to express my gratitude: to Professor Ron Elber for his encouragement and collaboration for my first research project at Cornell; to Dr Balaji Krishnapuram at Siemens for teaching me good work habits and time management in my first summer internship; and to Dr Sathiya Keerthi at Yahoo! Research for his interest in my development and discussions of research problems.

To my fellow PhD students I need to thank Dan Sheldon, Tudor Marian, and Kevin Walsh for being great officemates for many years. Thanks must also be given to fellow students in the Machine Learning Discussion Group

(MLDG) over the years for their friendships and collaboration, including Alex, Niculescu-Mizil, Filip Radlinski, Tom Finley, Benyah Shaparenko, Art Munson, Yisong Yue, Daria Sorokina, Nikos Karampatziakis and Ainur Yessenalina. Paper discussions and brainstorming half-baked ideas with them made studying and doing research in machine learning a much more fun experience.

I would also like to thank all the brothers and sisters in the Hong Kong Christian Fellowship and FICCC over the years for all their encouragements and prayers. Their love makes the winter in Ithaca warm. And thank God for His faithfulness and answering of my prayers, for the blessings I've received during these six years in Ithaca far exceed what I have asked or imagined.

Finally I need to thank my parents, my brother and my sister for their enduring patience and support over all these years when I am away from home, and for their believes in me in completing this degree. This thesis is dedicated to them.

The works presented in this thesis are supported by the grants NSF-Project IIS-0713483, NSF-Project IIS-0412894, and NSF-Project IIS-0905467.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

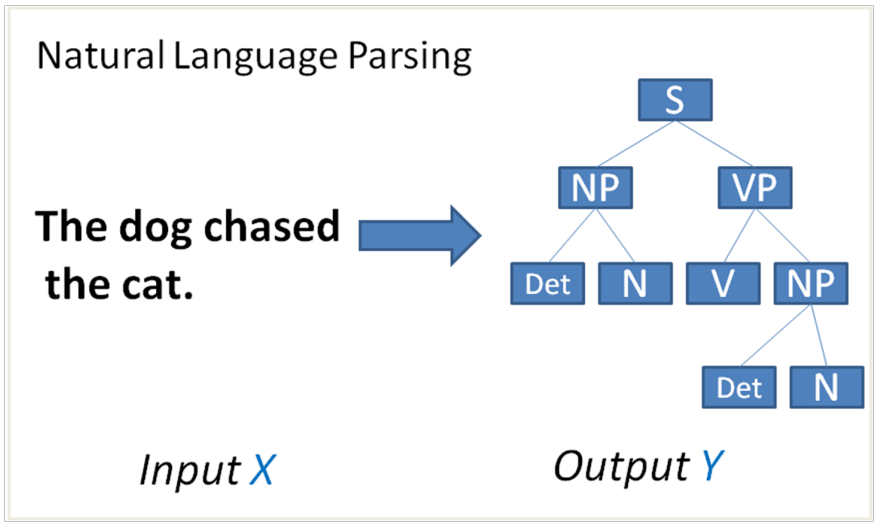
1.1 Structured Output Prediction

Structured output prediction is the problem of teaching machines to predict complex structured output objects such as sequences, trees, or graph matchings. It includes a very broad and diverse set of tasks, for example, parsing a Chinese sentence, aligning two protein sequences, segmenting a picture into people and objects, or translating a Japanese news article into English. It is very different from traditional tasks in machine learning and statistics such as classification or regression, where the output set $\mathcal{Y}$ contains only simple categorical labels or real values.

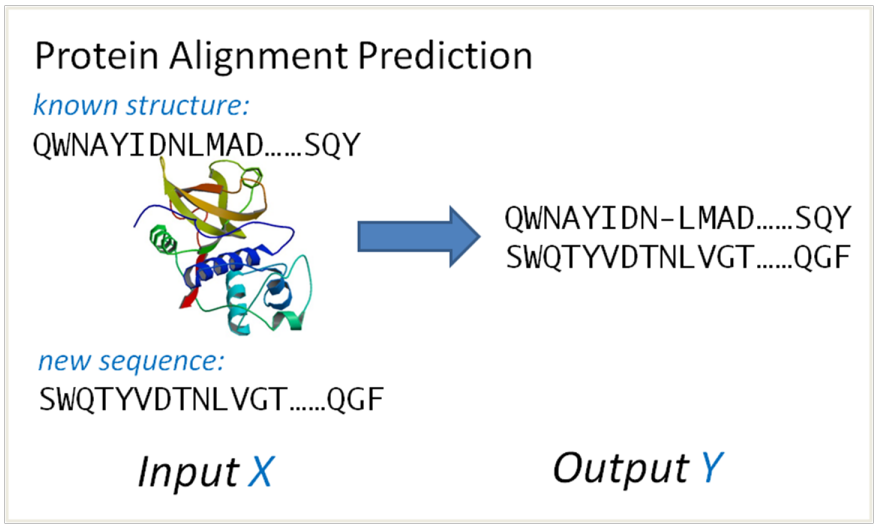
Figure 1.1 shows three specific structured output prediction tasks: parsing, protein sequence alignment, and object recognition. In natural language parsing (Figure 1.1(a)), the input space $\mathcal{X}$ is the set of well-formed English sentences, and the output space $\mathcal{Y}$ is the set of parse trees that show how the words are joined together to form meaningful sentences. In protein sequence alignment (Figure 1.1(b)), the input space $\mathcal{X}$ contains protein sequence pairs from a protein database, and the output space $\mathcal{Y}$ consists of alignments between these sequence-pairs that align structurally similar regions together. In object recognition (Figure 1.1(c)), the input space $\mathcal{X}$ can be different images taken inside offices, and the output space $\mathcal{Y}$ is the set of bounding boxes around monitors and keyboards inside the image.

What are the common features of these tasks, when compared against more

(a)



(b)



(c)

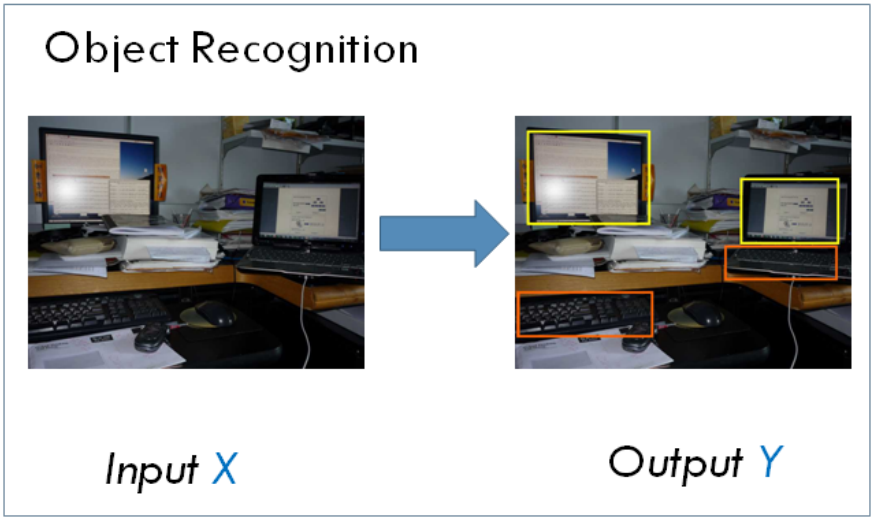


Figure 1.1: Three examples of structured output prediction problems

traditional machine learning problems such as classification or regression? The first observation is that these complex output objects usually consist of many correlated "parts". In Figure 1.1, the terminal node labels in the parse tree depends on the node labels of their parents. For example a verb phrase (VP) has to contain a verb (V) in its children. The alignment operations in protein sequence alignments also depend on the neighbouring alignment operations. For example, it is much more common to have long stretches of aligned amino acids or long stretches of unaligned gaps than to have short interleaving of aligned and unaligned positions. The neighbouring pixels in an image tend to belong to the same object, and different object-pairs can have certain spatial relations (e.g., it is quite common to locate a monitor above a keyboard). These local and global dependencies between the "parts" have to be taken into account when making predictions. These statistical dependencies are quite commonly, though not always necessarily, expressed using graphical models [70].

Another prominent common feature of these structured prediction tasks is the use of combinatorial optimization and search algorithms in prediction. For example, parsing can be performed using the CYK algorithm [72, 149], protein sequence alignments can be computed using the Smith-Waterman algorithm [118], which are both dynamic programming based algorithms. Image segmentation can be computed using graph cuts [16]. Once the local and global dependencies between the "parts" are fixed and the cost parameters determined, these combinatorial optimization algorithms help us to find out the highest scoring output parse tree, alignment, or segmentation. The problem of structured output learning is to learn the cost parameters so that the parse trees, alignments, or segmentations are as close to what we observe in the training set as possible. Thus it can be regarded in some way as the inverse problem of combi-

natorial optimization. We try to find out parameters given example structures, as opposed to finding highest scoring structures given fixed parameters.

1.2 Generative and Discriminative Models

We can state the structured output prediction problem more formally as follows. Given a set of training examples $S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \in (\mathcal{X} \times \mathcal{Y})^n$, we want to learn a prediction function $h : \mathcal{X} \to \mathcal{Y}$ in the following form:

$$h(x) = \text{argmax}_{y \in \mathcal{Y}} f_{\boldsymbol{\theta}}(x, y). \tag{1.1}$$

The function f is a scoring function that evaluates how well a particular output structure $y \in \mathcal{Y}$ matches an input $x \in \mathcal{X}$, and it is parameterized by the vector $\boldsymbol{\theta}$ that we want to learn. The argmax over all possible $y \in \mathcal{Y}$ extracts the highest scoring output as the prediction for x, and is usually computed using the combinatorial optimization algorithms such as the CYK algorithm or the Smith-Waterman algorithm previously mentioned. This is a very general setting that includes most learning problems for structured output prediction.

How should we learn the parameter vector $\boldsymbol{\theta}$ so that predictions that are close to what we observe in the training set can be reproduced? There are currently two main approaches for this. The first approach is called generative modeling while the second approach is called discriminative modeling.

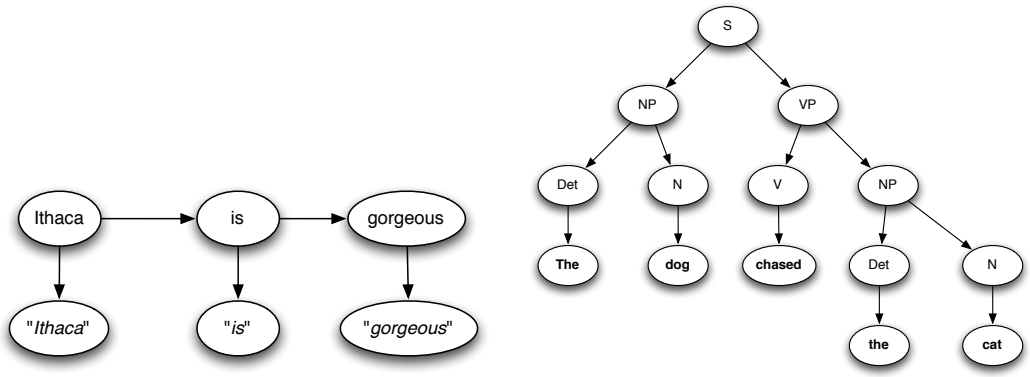


Figure 1.2: Examples of generative structures: HMM for speech recognition example (left) and parse tree (right)

1.2.1 Generative Models for Structured Output Predictions

Given an input x and an output y, generative model tries to estimate a joint distribution $P(x, y)$ of x and y. However this is usually a difficult problem to tackle directly given that both x and y are complex high-dimensional objects. The key idea here is to break down the complexity or high dimensionality by explicitly stating how the complex input-output pair (x, y) are generated by composing simple parts, and thus the name *generative* modeling. This is also equivalent to stating how the complex distribution $P(x, y)$ factorizes into a product of simpler distributions on the parts in x, y.

The classic Hidden Markov Model (HMM) [105] is a good example of generative modeling, and it was very widely applied to the problem of speech recognition in the early days. In speech recognition (a simplified version of the problem) the input x (called observations) is a sequence of sounds in human speech, and the output y (called hidden states) is a corresponding set of words in the English language. Hidden Markov models make the assumption that the observations at time step t are generated by the hidden state at time step t only,

and the hidden state at time step t depends only on the previous hidden state at time step $t-1$. Thus to generate English speech for the sentence "Ithaca is gorgeous", we would first generate the word "Ithaca" randomly from the initial distribution, and then generate a pronunciation of "Ithaca" from our emission distribution. Next we generate a second English word conditioned on the previous word "Ithaca", and suppose the word is "is", and after that we generate a pronunciation of "is" using the emission distribution. And we repeat the same for the third word "gorgeous". Thus the process of generating human speech can be modeled in a very simple linear fashion (Figure 1.2, left).

As another example let us look at the parse tree of the sentence "The dog chased the cat" in Figure 1.2. We can imagine there is a stochastic process (stochastic context free grammar) that starts from the root of the tree as a whole sentence (S), and then split into a pair of noun phrase (NP) and verb phrase (VP). The noun phrase then splits into a determiner ("the") and a noun ("dog"), and the verb phrase further splits into a verb and a noun phrase that gives the whole subtree for the expression "chases the cat". At each node we generate a leaf node or an internal node conditioned on the previous splits in the tree according to a local conditional probability distribution. For different runs we could have generated alternative sentences and parse trees such as "The cat chased the dog", or "The dog chased the fat cat" in these tree growing processes.

For both the HMM and parse tree generation example the observed input-output pair (x, y) are generated sequentially, part by part, through simple stochastic generative processes. These processes are completely specified by local conditional probability tables that govern the decision at each step. To put it more precisely, we estimate our scoring function f by taking it to be the joint

probability (or equivalently its logarithm):

$$\begin{aligned} f_{\boldsymbol{\theta}}(x, y) &= P_{\boldsymbol{\theta}}(x, y) \\ &= \prod_i P(y_i \mid \pi(y_i)) \prod_j P(x_j \mid \pi(x_j)), \end{aligned}$$

where x_j, y_i are the observed and hidden nodes in the directed acyclic graphs in Figure 1.2. The notation $\pi(u)$ refers to the parent node of u in the directed acyclic graph. The local conditional probabilities for transitions between hidden nodes $P(y_i \mid \pi(y_i))$ and emissions for observed nodes $P(x_j \mid \pi(x_j))$ are the parameters in the vector $\boldsymbol{\theta}$.

Since the complex process of jointly generating a speech sequence and an English sentence (or a parse tree and an English sentence in the other example) is broken down into a series of local decisions, parameter estimation of $\boldsymbol{\theta}$ through maximum likelihood becomes easy as in most cases it boils down to counting.

1.2.2 Discriminative Models for Structured Output Predictions

The second approach is called discriminative modeling, which takes a completely different view on structured output prediction. The training sample we are given comes in the form of input-output pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$. Generative modeling tries to model all the observations by estimating a joint distribution $P(x, y)$. However if we think carefully this is not exactly the task that we are given. Consider a hypothetical extreme case when there is one possible output $y_0 \in \mathcal{Y}$, then no matter what the input x is all we can do is to predict y_0. It will be a huge waste of effort to model the joint distribution $P(x, y)$ (which is essentially $P(x)$) in this case, since the input x can have arbitrarily complex distribution but it would not help us in the prediction of y at all. Even not in this extreme sit-

uation it is quite common to have information in input x that is irrelevant for the prediction of y. For example, when we are building a generative model for detecting whether there are humans in a picture, we could be spending a lot of effort to model all the clutter and non-human objects in the background in order to estimate a good joint distribution $P(x, y)$.

A second major difficulty in applying generative models in practice is that they are difficult to design. While generative models for the sequence tagging and parse tree examples are relatively straightforward and reasonable, in many structured output prediction problems it is very hard to design a generative model that explains all the observed features in input x well. In the human-detection example just mentioned it is almost impossible to describe a good generative process that models all the clutter and background objects on pictures with humans on it. In machine translation it is also difficult to describe a generative model that transforms a French sentence into an English sentence. Many simplifying assumptions have to be made [17], such as each word being translated independently or the limited re-arrangement when translating from French into English. In the protein alignment example that we are going to discuss in Chapter 4, it is not easy to decide whether to assume (which is obviously wrong biologically) that the amino acid identity at an aligned position is independent of their secondary structures to simplify the parameter estimation. These modeling decisions are really hard to make and worse still we might have to re-design the model structures whenever we obtain new observed features.

Discriminative modeling tries to sidestep all these problems in generative modeling by tackling the estimation of the scoring function f in Equation (1.1) directly. It tries to learn a function f such that good output structures score

higher than bad output structures for a fixed input x, and hence the name *discriminative* modeling, because the function f helps us to differentiate between good and bad output structures. No attempt is made on modeling the input distribution for x or the joint distribution $P(x, y)$, and thus a lot of the difficult modeling decisions can be avoided. One common choice is to assume f to be linear, with

$$f_{\boldsymbol{\theta}}(x, y) = \boldsymbol{\theta} \cdot \boldsymbol{\Phi}(x, y),$$

where $\boldsymbol{\Phi}(x, y)$ is a high dimensional feature vector that extracts features which indicate whether the input-output pair (x, y) is a good fit or not.

Conditional random field (CRF) [81], a discriminative structured output learning method, finds the parameter vector $\boldsymbol{\theta}$ and hence f by minimizing the following objective:

$$\min_{\boldsymbol{\theta}} \frac{1}{2}\|\boldsymbol{\theta}\|^2 - C\sum_{i=1}^{n}[\boldsymbol{\theta} \cdot \boldsymbol{\Phi}(x_i, y_i) - \log\left(\sum_{\hat{y}\in\mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{\Phi}(x_i, \hat{y}))\right)]. \tag{1.2}$$

Structural support vector machine (Structural SVM) [132], which is the topic of this thesis, finds f by minimizing the following objective:

$$\min_{\boldsymbol{\theta}} \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C\sum_{i=1}^{n} \max_{\hat{y}\in\mathcal{Y}}[\boldsymbol{\theta} \cdot \boldsymbol{\Phi}(x_i, \hat{y}) - \boldsymbol{\theta} \cdot \boldsymbol{\Phi}(x_i, y_i) + \Delta(y_i, \hat{y})], \tag{1.3}$$

where Δ is a loss function measuring how different the output $\hat{y}$ is when compared against the reference output y_i.

We shall have a more in-depth discussion of these methods in the next chapter. Ignoring the regularization term $1/2\|\boldsymbol{\theta}\|^2$ for the moment, we can see that both CRFs and structural SVMs try to set the score of the correct output y_i in the training set higher than alternative outputs $\hat{y} \in \mathcal{Y}$, though through different ways. CRFs acheive this by maximizing the conditonal likelihood $P(y \mid x)$, trying to make the score of the correct output $\boldsymbol{\theta} \cdot \boldsymbol{\Phi}(x_i, y_i)$ large with respect to the

log-sum of all other outputs $\log(\sum_{\hat{y}\in\mathcal{Y}} \exp(\boldsymbol{\theta}\cdot\boldsymbol{\Phi}(x_i,\hat{y})))$. Structural SVMs achieve this by maximizing the score difference between the correct output $\boldsymbol{\theta}\cdot\boldsymbol{\Phi}(x_i,y_i)$ and the scores of all other outputs $\boldsymbol{\theta}\cdot\boldsymbol{\Phi}(x_i,\hat{y})$, with the required margin enforced by a loss function Δ.

We can see in both of these objectives efforts are made to discriminate the correct output y_i against all alternative outputs $\hat{y}$ that are not observed, while no effort is made to model the input distribution for x_i (or any alternative inputs). As far as the minimization objective is concerned all the inputs x_i are fixed. This is a common feature of discriminative modeling, and can be seen as a more pragmatic approach compared to the more ambitious approach of joint density estimation in generative modeling.

1.3 Structural Support Vector Machines

Structural support vector machine (Structural SVM), the focus of this thesis, is a very successful and popular algorithm for discriminative structured output learning. First as a discriminative learning algorithm it allows flexible construction of features to improve prediction accuracy. As in Equation (1.3) the distribution of input x_i is not explicitly modeled, allowing us to construct complex features based on the input x_i in the joint feature vector $\Phi(x,y)$ without modifying the structure of the optimization problem. This particular flexibility and ease to incorporate new features to improve prediction performance are the main reasons for the popularity of discriminative models such as structural SVMs and CRFs in recent years.

Secondly although discriminative modeling gives us the freedom to con-

struct rich features from the input x to improve prediction accuracy, it also becomes much easier to overfit to the training data when we have the power to incorporate these high dimensional input features. Structural SVM, like the binary support vector machine for classification, controls overfitting by maximizing the margin between alternative outputs. The influential work of Vapnik [133] on regularized risk minimization laid the foundation for discriminative modeling, where the importance of capacity control (regularization) is emphasized and linked to the generalization ability of learning algorithms. Regularization is particularly important when we have high dimensional feature space and small training sample sizes. The implementation of this idea in the form of Support Vector Machines (SVM) gives rise to one of the most popular machine learning algorithm in the last decade [34], with high classification accuracies on many applications such as text classification [62], object recognition [103], classification of gene expression data [56], and countless many other applications. Binary SVMs work by constructing a large-margin hyperplane in high dimensional feature space as a separator between positive and negative examples. Structural SVM is a generalization of binary SVMs for structured output learning. It inherits many of the attractive properties of SVMs, such as the ability to explicitly control margins between different outputs, the ability to incorporate high dimensional input features without overfitting, and the ability to incorporate nonlinear kernels.

Thirdly compared to CRFs, structural SVMs do not require the computation of log-sum over all possible output structures (called the partition function) in Equation (1.2). This can be beneficial in certain cases such as learning matchings or spanning trees, when computing the sum over all possible matchings or spanning trees have a higher computational complexity than computing the

highest scoring output structure. The explicit use of a loss function Δ to control the margins also helps us tune the parameter vector in different application settings. Because of all these advantages structural SVMs have been applied to many different applications in diverse domains, including natural language processing, computational biology, information retrieval, computer vision, etc. We will discuss some of these applications in greater detail in the later chapters.

1.4 Organization and Contributions of this Thesis

This thesis presents some new results on discriminative structured output learning based on structural SVMs.

1.4.1 Structural SVM for Protein Alignments

Protein sequence alignment is an important step in the homology modeling approach for solving protein structures, but it is also a very hard problem for sequence pairs with low sequence similarity but high structural similarity. There are a lot of relevant information for determining the alignment such as amino acid residue, secondary structures, water accessibility, and evolutionary information through PSI-BLAST profiles in databases for proteins with known structures. This is an ideal application for discriminative structured output learning since we can incorporate all these rich features without having to model their dependencies. In this thesis we apply the structural SVM algorithm to learn protein alignment models incorporating these rich features over a large dataset. Experiments show that it outperforms some of the state-of-art generative align-

ment models on large test sets and is practical to train. This application also illustrates many of the design decisions in applying structural SVM for structured output learning such as the design of feature maps and loss functions, and also choosing a corresponding efficient inference procedure for training.

1.4.2 Latent Structural SVM

Despite the success of structural SVMs in many different applications, there are still issues that hinder their even wider adoption. One important issue is their inability to handle latent variables or missing information. In many applications in natural language processing and computer vision some of the crucial modeling information is not present in the training data, and yet it is important to include it to learn models of good predictive performance. For example in object or human recognition, it is important to know the positions of the parts of an object to accurately recognize the object, such as the positions of the wheels in a car or the positions of arms and legs of a pedestrian. And yet as labels we are only given a rough bounding box on where the target object is without any information on the parts. As another example consider the training data we usually have in machine translation, which are sentence-pairs in the source language (say French) and the target langauge (say English). However, it will be really useful to know roughly which French word corresponds to which English word in the sentence pair, but such information is usually not available. In this thesis we give the first natural extension of the structural SVM algorithm to handle latent variables. This greatly increases the scope of application of the structural SVM algorithm to new problems, and also allows us to solve old problems in new formulations. We give efficient training algorithm for our

latent structural SVM formulation, and demonstrate how it can be applied via three different applications in computational biology, natural language processing, and information retrieval.

1.4.3 Training Nonlinear Structural SVMs with Kernels using Approximate Cutting Plane Models

The ability to produce nonlinear decision boundaries through the use of kernels was one of the main reasons of the wide applicability and popularity of support vector machines in classification. Using kernels such as the polynomial kernels and the Gaussian kernel, SVM is one of the best off-the-shelf classification algorithms when applied to a new classification problem without any tuning. The use of nonlinear kernels in structured output learning is equally beneficial since it allows the learned function for scoring different output structures to become nonlinear. However, the cost of training nonlinear SVMs with kernels is much higher than training linear SVMs because the target function we learn is represented implicitly as a sum of kernel functions evaluated at a large number of training points. There are many works on approximation in training binary classification SVMs with kernels [145, 46, 131, 14], but they are either not directly applicable to structural SVMs without significant modification or they do not give good enough tradeoffs between approximation quality and training costs. In this thesis we propose to directly modify the cutting plane algorithm for the approximation. The cutting plane algorithm is one of the most successful algorithm for training support vector machines and structural support vector machines. We will explore approximating each cutting plane generated

by the cutting plane algorithm so that the cost of kernel function evaluations is reduced. In particular we are going to look at two different approximation schemes, one using random sampling and the other using a preimage optimization approach. Our proposed algorithms result in improved training time by reducing the number of kernel evaluations required while maintaining accuracies close to the exact algorithm, and the preimage approach also has better accuracy-sparsity tradeoff when compared against other approximation algorithms on binary classification. Moreover they can be easily extended to handle the use of nonlinear feature functions in structural SVMs.

1.4.4 Thesis Organization

This thesis is organized as follows. In Chapter 2 we give an introduction to discriminative training of structured output prediction models, with an emphasis on structural SVMs.

In Chapter 3 we describe in detail the cutting plane algorithm for solving the optimization problems associated with structural SVMs. It provides the basis for the various algorithms discussed in the following chapters.

In Chapter 4 we illustrate the benefits and challenges of discriminative training through our work on using structural SVMs to train alignment models for protein sequence alignments.

In Chapter 5 we describe our work on extending structural SVMs to handle latent variables, which greatly extends their scope of applications.

In Chapter 6 we describe our work on improving the training efficiency of

structural SVMs with approximate cutting plane models, specifically for the cases where nonlinear kernels are used.

Chapter 7 contains our concluding remarks and points out some interesting future research directions.

CHAPTER 2

STRUCTURAL SUPPORT VECTOR MACHINES

2.1 A Part-Of-Speech Tagging Example

In this chapter we are going to introduce Structural Support Vector Machines (Structural SVMs) and see how we can use it for predicting structured outputs. For this purpose let us use a very simple example to see how we can do structured output prediction with traditional generative modeling, and then see how it can be modeled discriminatively with structural SVMs. The example application that we are going to consider is part-of-speech tagging (POS tagging) in natural language processing.

Given an English sentence, part-of-speech tagging tries to determine each word in the sentence to see whether it is a noun, a verb, an adjective, a determiner, etc. It is not a trivial task as many English words can have more than one part-of-speech (e.g., "bank" as verb and "bank" as noun). It is a very basic low-level task in natural language processing that supports many other higher level tasks such as question-answering, sentiment analysis, etc. For example, knowing where the nouns are can help us recognize what people or organizations are involved, knowing the verb can help us understand what is happening, and knowing the adjective can help us decide the speaker's opinion or sentiment. It is a structured-output prediction task as the part-of-speech tag at each position in the sentence is highly correlated with the POS tags at the positions immediately before and after it.

One of the earliest generative models for modeling and predicting sequence

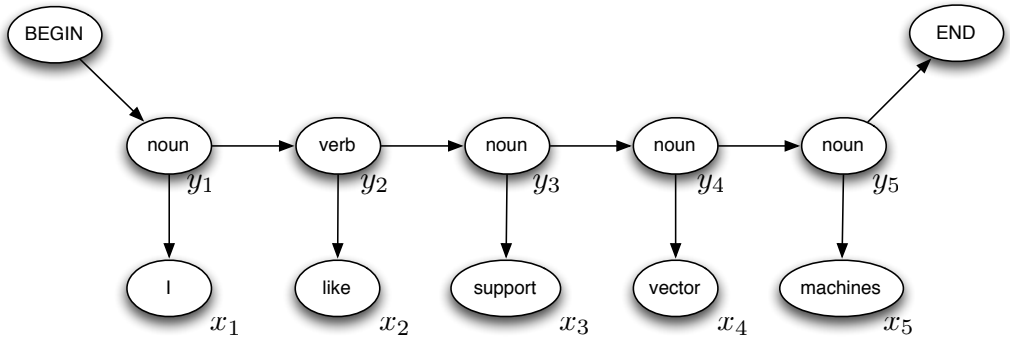


Figure 2.1: HMM example

data is Hidden Markov Model (HMM) [105]. Hidden Markov models estimates the joint distribtion $P(x, y)$ between the input $x = (x_1, x_2, \ldots, x_L)$ (called the observation sequence) and the output $y = (y_1, y_2, \ldots, y_L)$ (called the hidden state sequence), where L is the length of the sequence. It tells us how an (x, y) sequence pair can be generated one position after another by making stochastic local decisions. Consider the sentence "I like support vector machines" in Figure 2.1, with the correct part-of-speech tags "Noun Verb Noun Noun Noun". The outputs y_i are the part-of-speech tags, and they are also called hidden states in HMM terminology as they are not observed and are the labels that we want to predict. The inputs x_i are the English words, and they are called the observations. The arrows in the figure indicate dependency relations, and we can see that each hidden state y_i only depends on the immediate hidden state y_{i-1} before it. This is the Markov assumption and hence the name hidden Markov models. Also, each observation x_i also only depends on the hidden state y_i at the same position.

Thus we can generate the whole (x, y) pair by making sequential local decisions. We can generate the first hidden state y_1 by throwing a dice according

Table 2.1: Conditional Probabilty Tables for HMM example

		y_{i+1}				
	$P(y_{i+1} \mid y_i)$	N	V	Adj	Det	END
	N	0.3	0.5	0.05	0.15	0.1
	V	0.4	0.2	0.3	0.1	0.1
y_i	Adj	0.6	0.05	0.2	0.15	0.1
	Det	0.6	0.05	0.3	0.05	0.1
	BEGIN	0.25	0.25	0.25	0.25	0

		x_i											
	$P(x_i \mid y_i)$	...	I	...	Like	...	Machine	...	Support	...	Vector	...
	N	...	0.05	...	0	...	0.001	...	0.005	...	0.001	...
y_i	V	...	0	...	0.02	...	0	...	0.01	...	0	...
	Adj	...	0	...	0.0001	...	0	...	0	...	0	...
	Det	...	0	...	0	...	0	...	0	...	0	...

to the conditional probability tables in Table 2.1 starting from the special initial state "BEGIN", and suppose it comes up to be "Noun". Then we generate the first observation x_1 conditioned on y_1 by throwing a dice according to the emission table, and suppose it comes up with "I". Then we continue to generate stochastically the second hidden state y_2 conditioned on y_1, and suppose it comes up to be a "Verb". We continue this stochastic generative process by following the arrows and looking up the conditional probability tables, until the whole sentence "I like support vector machines" has been generated. Then finally the "END" state is generated from y_4 and we stop the whole generative process. This generative process is stochastic and assigns a probability to every (x, y) pair. For example, if our dice throws come out differently we might

end up having the sentence ”I like hidden Markov models” instead. The joint probability $P(x, y)$ is just the product of all the local conditional probabilities:

$$P(x, y) = P(y_1)P(x_1 \mid y_1) \prod_{i=2}^{L} P(y_i \mid y_{i-1})P(x_i \mid y_i). \quad (2.1)$$

In our example the probability of the pair (”I like support vector machines”, ”Noun Verb Noun Noun Noun”) is

$$0.25 \times 0.05 \times 0.5 \times 0.02 \times 0.03 \times 0.005 \times 0.3 \times 0.001 \times 0.3 \times 0.001 \times 0.1 = 1.6875 \times 10^{-15}$$

by multiplying all the local probabilities on the arrows in the graph.

Given all the local probability tables and an input sequence x, we would like to find out the output sequence y that maximizes the conditional probability (or equivalently joint probability) during prediction:

$$\text{argmax}_{y \in \mathcal{Y}} P(x, y) = \text{argmax}_{y \in \mathcal{Y}} P(y_1)P(x_1 \mid y_1) \prod_{i=2}^{L} P(y_i \mid y_{i-1})P(x_i \mid y_i). \quad (2.2)$$

This can be done by the Viterbi algorithm [138], which is essentially a dynamic programming algorithm that recursively computes the maximum output subsequence from y_1 to y_i in a left-to-right manner, for $1 \leq i \leq L$.

We are mostly interested in the learning problem or the inverse problem of estimating the local conditional probability tables given a set of training examples $\{(x_i, y_i)\}_{i=1}^{n}$ (we are overloading the notation for the ith training example x_i with the i observation of an input sequence x, but the meaning should be clear from context). In hidden Markov models these tables are usually estimated via maximizing the joint likelihood $P(x, y)$ over the whole training sample $\{(x_i, y_i)\}_{i=1}^{n}$. In the absence of missing values this will simplify to counting. For example, to estimate $P(y_{i+1} = \text{”}Verb\text{”} \mid y_i = \text{”}Noun\text{”})$, we can just count the number of times a ”Verb” follows a ”Noun” in the whole training sample, and divide the number by the total number of ”Noun”s in the training sample.

Generative modeling tries to build a complete model $P(x, y)$ on the observed data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, and one advantage of this is we can generate new example pairs (x, y) that "look like" the observed data. For example, Shannon used a Markov model to generate random English sentences in his landmark paper *A Mathematical Theory of Communication* [115]. A large body of work in the area of *language modeling* in natural language processing is on building better and better generative models of English that gives improved approximation of true English sentences [107]. Fitting a generative model to the data and diagnosing how well they match can give us a better understanding of the observed data. For example by moving from a simple Markov chain to stochastic context free grammar we can build more accurate models of English sentences, because the stochastic context free grammar can capture more non-local dependencies in true English sentences. Building generative models is an intricate process that requires a balance between simplicity and accuracy of the model, and in most cases the computational cost of estimating and applying the model as well. Therefore designing good generative models require a lot of domain expertise on the problem. Generative modeling is by no means restricted to linear sequences such as English sentences, but include many other structures such as trees [29], objects and shapes [44], and random graphs [96, 94] in social networks as well.

In contrast to generative modeling, discriminative modeling does not try to build a full model of the observed training data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, but focuses on the prediction task at hand. Since our task is to predict an output y given a fixed input x, there is no need to model the whole joint distribution $P(x, y)$. For example, an accurate estimation of the conditional distribution $P(y \mid x)$ is sufficient for prediction. This problem is particularly eminent in

classification, where it is common to have very high dimensional feature vector $\boldsymbol{x}$. These high dimensional feature vectors make it difficult to model all the correlations of the features in a generative manner. On the other hand the output space $\mathcal{Y}$ is very simple (binary or a few classes), making the task of modeling the conditional distribution $P(y \mid x)$ much easier. Indeed Vapnik [133] believes that using density estimation to solve prediction problems is unnecessary and this is just like trying to solve a simpler problem (prediction) via a much harder intermediate problem (density estimation). The Empirical Risk Minimization (ERM) principle that he proposed tries to directly minimize the prediction risk or loss on the training data while restricting the complexity of the function classes used through regularization to prevent overfitting. This is a form of discriminative modeling, and forms the basis of support vector machines and structural support vector machines.

Before going on to discuss how we can model this POS tagging problem discriminatively with structural SVMs, let us take a diversion and look at binary classification SVMs and see how it can be generalized to structural SVMs.

2.2 Support Vector Machines

In learning a classifier for binary classification we are usually given a set of training examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$. The input features $\boldsymbol{x}_i \in \mathbb{R}^d$ are usually d-dimensional vectors describing the properties of the input example, and the labels $y_i \in \{+1, -1\}$ are the response variables/outputs we want to predict. For example, our task could be to predict whether it is going to rain tomorrow in Ithaca based on various measurements collected today. The label $+1$ indicates

that it is going to rain, and -1 indicates that it is not going to rain. The input features could be continuous measurements such as local atmospheric pressure, average/minimum/maximum temperatures, average humidity, or categorical features such as month and date in the year. Let us assume that there are d of such features (after suitable binarization of the categorical features). Suppose we have collected such data for the past year ($n = 365$), then we can regard these training examples as 365 points in a d-dimensional vector space, with labels $+1$ or -1 over them. The concept of Support Vector Machines (SVM) as a binary classification algorithm is very simple: construct the most "stable" linear hyperplane in this d-dimensional space that separates the positive and negative examples (see Figure 2.2). This most stable hyperplane can be found by solving the following quadratic optimization problem:

Optimization Problem 2.1. (HARD-MARGIN BINARY SVM (PRIMAL))

$$\begin{aligned} &\min_{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &s.t.\ \forall i, y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \geq 1 \end{aligned} \tag{2.3}$$

The weight vector $\boldsymbol{w}$ is the normal of the hyperplane, and the scalar b is the offset of the hyperplane from the origin in the vector space. Note that the (signed) distance of the point $\boldsymbol{x}_i$ from the hyperplane is $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\cdot\boldsymbol{x}_i + \frac{b}{\|\boldsymbol{w}\|}$ (projection onto unit normal). Therefore the constraint $y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i+b) \geq 1$ is essentially declaring that all the examples have to lie on the correct side of the hyperplane, with a minimum distance of $1/\|\boldsymbol{w}\|$. Thus by minimizing the norm of $\boldsymbol{w}$ in the objective, we are effectively maximizing the minimum distance to the hyperplane over all examples (called the *margin*).

However, for most of the training sets we receive for various classification problems, there do not exist any linear separators that have all positive exam-

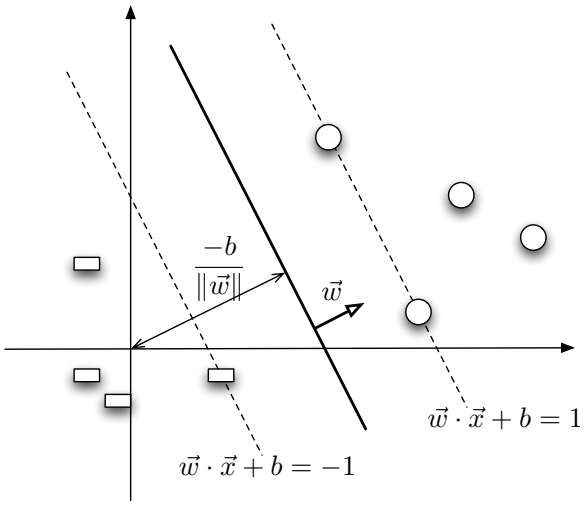


Figure 2.2: Binary Support Vector Machines

ples on one side and all negative examples on the other. This could be due to noise in the measurements, or simply that the features observed are not sufficient for confident prediction/separation of the two classes. In these cases the original quadratic optimization problems no longer have any feasible solutions, and we need to relax the linear constraints to allow feasible solutions. One way to do this is to introduce a non-negative slack variable ξ_i for each example, to allow the linear constraint to be violated by the amount ξ_i (see Figure 2.3). The optimization problem then becomes:

Optimization Problem 2.2. (SOFT-MARGIN BINARY SVM (PRIMAL))

$$\begin{aligned} \min_{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^n} & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ s.t.\ & \forall i, y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \geq 1 - \xi_i \\ & \forall i, \xi_i \geq 0 \end{aligned} \tag{2.4}$$

The parameter C controls the trade-off between the two conflicting goals of having a larger margin and that of having smaller errors on the training set

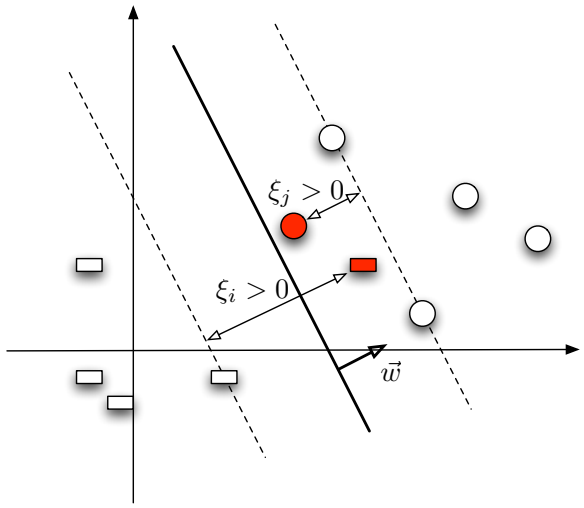


Figure 2.3: Binary Support Vector Machine with Slack Variables

(sum of slacks). This is an important parameter for us to control overfitting to the training data.

To every minimization problem there is usually a closely related maximization problem called the dual problem (and the original one is called the primal). The values of the dual problem can be regarded as lower bounds of the values of the primal problem. For most convex optimization problems under mild conditions (for example Slater's constraint qualification or when the inequalities are all affine [15]) the value of the optimal primal solution and the value of the optimal dual solution coincide, a property which we call *strong duality*. In those cases one can solve the dual optimization problem and recovers the primal solution from that. The optimization problem for support vector machine is one such example where we can solve the dual problem. The dual optimization problem looks like the following:

Optimization Problem 2.3. (BINARY LINEAR SVM (DUAL))

$$\begin{aligned} &\max_{\boldsymbol{\alpha}\in\mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j \\ &s.t.\ \forall i, 0 \leq \alpha_i \leq C \\ &\sum_{i=1}^{n} y_i\alpha_i = 0 \end{aligned} \tag{2.5}$$

The variables α_i are called the dual variables, and each α_i corresponds to exactly one constraint $y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \geq 1 - \xi_i$ in the primal. The primal and dual solution are related by:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i, \tag{2.6}$$

and thus we can easily recover the weight vector $\boldsymbol{w}$ after knowing all the α_i's.

Also the primal and dual variables have to satisfy the following *complementary slackness* condition:

$$\forall i, \alpha_i[1 - \xi_i - y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b)] = 0.$$

One consequence of the complementary slackness condition is that when the constraint $y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i+b) > 1$ is satisfied strictly so that the slack variable $\xi_i = 0$, the corresponding dual variable α_i has to be equal to 0. This is because $[1-\xi_i-y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b)] < 0$ in this case and α_i is forced to be 0 by the complementary slackness condition. By Equation (2.6) examples $(\boldsymbol{x}_i, y_i)$ with dual variable $\alpha_i = 0$ have no effect in determining the expansion of weight vector $\boldsymbol{w}$, thus the solution of the optimization problem will remain the same if those examples are removed from the training set (examples lying outside the dotted line in Figure 2.3). Only those examples with dual variables $\alpha_i > 0$ can affect the solution of the SVM optimization problem. These examples are called *support vectors*, since they are like vectors exerting forces pushing towards (or supporting) the hyperplane,

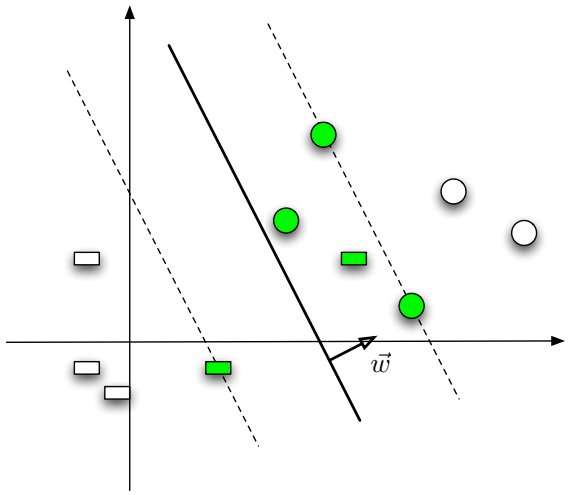


Figure 2.4: Binary support vector Machine with slack variables, with support vectors highlighted

and hence the name support vector machines. All the support vectors for the same problem as in Figure 2.3 are highlighted in Figure 2.4.

An interesting observation about the dual optimization problem in the above is that the only place where the input features enter the problem is through the inner products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$. Vapnik and his colleagues, building on previous works by [2], had the insight that these linear inner products can be replaced by more general functions on $\mathcal{X} \times \mathcal{X}$ called kernel functions to build complex nonlinear classification boundaries. The following is the dual SVM optimization problem with kernel functions:

Optimization Problem 2.4. (BINARY SVM WITH KERNELS (DUAL))

$$\begin{aligned} &\max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &s.t.\ \forall i, 0 \leq \alpha_i \leq C \\ &\sum_{i=1}^{n} y_i \alpha_i = 0 \end{aligned} \tag{2.7}$$

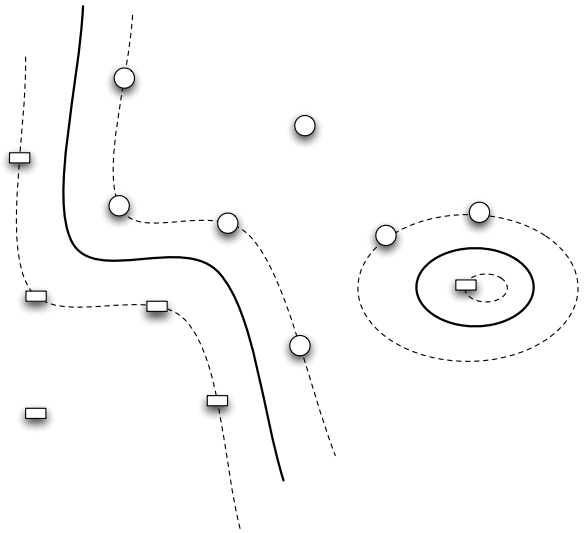


Figure 2.5: Binary SVM with Gaussian Kernels. The solid line is the decision boundary, while the dotted lines are $\boldsymbol{w} \cdot \boldsymbol{x} = \pm 1$. Notice the ability to introduce nonlinear and even disjoint decision regions using the Gaussian kernel.

Kernel functions are very general in the sense that they only need to satisfy a few conditions such as positive-semidefiniteness so that they can act like inner products for elements in $\mathcal{X}$, but in a much higher and possibly infinite-dimensional space. Figure 2.5 shows the decision boundary with Gaussian kernels (or radial basis function kernel). Notice the nonlinearity of the decision boundary and also the ability have disjoint regions of the same class.

There is a whole body of theory on kernels in terms of Reproducing Kernel Hilbert Spaces [111], and numerous applications in areas such as computer vision [52, 148] and computational biology [112]. We will have a longer discussion on kernels in Chapter 6 when we talk about the use of nonlinear kernels in structural SVMs.

2.3 From Binary to Multi-Class Support Vector Machines

After a brief introduction to SVM let us consider the problem of how it might be extended for structured output prediction. For simplicity let us first assume we are working with a linear feature space.

The binary support vector machine is a very geometrically intuitive classification algorithm. Since there are only two possible output classes, finding the most stable hyperplane that separates the two classes as our decision boundary is a very reasonable idea. However, what should we do when the output y is no longer binary, but contains multiple (> 2) different categories, or even exponentially many possible output classes as in structured output prediction?

Consider the case for classification when there is more than two output classes. It is no longer possible to have one separating hyperplane $\boldsymbol{w}$ that cleanly puts all examples in one class with $\boldsymbol{w} \cdot \boldsymbol{x} > 0$ and all other examples in the other class with $\boldsymbol{w} \cdot \boldsymbol{x} < 0$, since there are more than two classes (we are dropping the bias term b here for simplicity, and the effect of the bias term can be simulated by adding an extra constant feature in the examples $\boldsymbol{x}_i$). Clearly to describe the decision boundary we need more than one hyperplane $\boldsymbol{w} \in \mathbb{R}^d$, and it is not clear what would be the most natural generalization of binary SVM.

Let us generalize from what we learn in the binary case. In general we have a linear decision score $\boldsymbol{w} \cdot \boldsymbol{x}$ for each example $\boldsymbol{x}$. The more positive the score is, the more likely will the example belong to the positive class; likewise the more negative the score is, the more likely for the example to belong to the negative class. Suppose there are k classes, we can set up k weight vectors such that the

decision score for the ith class is:

$$\boldsymbol{w}_i \cdot \boldsymbol{x}$$

for $1 \leq i \leq k$. To find out the most likely class for example $\boldsymbol{x}$, we compute

$$\text{argmax}_{1 \leq i \leq k} \boldsymbol{w}_i \cdot \boldsymbol{x}.$$

The original binary SVM ($k = 2$) can be recovered by having the extra constraint $\boldsymbol{w}_1 = -\boldsymbol{w}_2$ to remove the extra degree of freedom.

Here we're going to consider the multi-class SVM generalization by Crammer & Singer [35], since it poses the training problem as a single optimization problem without breaking the classification problem down into a series of binary decision problems [106].

Optimization Problem 2.5. (MULTI-CLASS SVM (CRAMMER & SINGER))

$$\begin{aligned} \min_{\boldsymbol{w}_c \in \mathbb{R}^d, \boldsymbol{\xi} \in \mathbb{R}^n} & \sum_{c=1}^{k} \frac{1}{2} \|\boldsymbol{w}_c\|^2 + C \sum_{i=1}^{n} \xi_i \\ s.t.\ & \forall i, \forall c \neq y_i \\ & \boldsymbol{w}_{y_i} \cdot \boldsymbol{x}_i - \boldsymbol{w}_c \cdot \boldsymbol{x}_i \geq 1 - \xi_i \\ & \forall i, \xi_i \geq 0 \end{aligned} \tag{2.8}$$

We can notice that the objective function is essentially the same as binary SVM. The only difference comes from the constraints, which essentially says that the score of the correct label $\boldsymbol{w}_{y_i} \cdot \boldsymbol{x}$ has to be greater than the score of any other classes $\boldsymbol{w}_c \cdot \boldsymbol{x}$, so there are $k - 1$ constraints in total when there are k classes. There is one slack variable ξ_i for each example, shared among the $k - 1$ constraints.

We can simplify the notation by stacking the class weight vectors to form a single weight vector $\boldsymbol{w}$ and introducing the notation for joint feature vector

$\boldsymbol{\Phi}(\boldsymbol{x}, y)$:

$$\boldsymbol{w} = \begin{pmatrix} \boldsymbol{w}_1 \\ \boldsymbol{w}_2 \\ \vdots \\ \boldsymbol{w}_{k-1} \\ \boldsymbol{w}_k \end{pmatrix}, \boldsymbol{\Phi}(\boldsymbol{x}, y) = \begin{pmatrix} \mathbf{0} \\ \vdots \\ \boldsymbol{x} \\ \vdots \\ \mathbf{0} \end{pmatrix},$$

where $\boldsymbol{x}$ appears in the yth block in $\boldsymbol{\Phi}(\boldsymbol{x}, y)$. Then the whole opimization problem can be written succinctly as:

Optimization Problem 2.6. (MULTI-CLASS SVM (CRAMMER & SINGER))

$$\begin{aligned} \min_{\boldsymbol{w} \in \mathbb{R}^{kd}, \boldsymbol{\xi} \in \mathbb{R}^n} & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i \\ s.t.\ & \forall i, \forall c \neq y_i \\ & \boldsymbol{w} \cdot \boldsymbol{\Phi}(\boldsymbol{x}_i, y_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(\boldsymbol{x}, c) \geq 1 - \xi_i \\ & \forall i, \xi_i \geq 0 \end{aligned}$$

2.4 From Multi-class to Structural Support Vector Machines

Structured output prediction problem, in its simplest form, can be regarded as a multi-class prediction problem, abeit with a huge number of classes in the output space $\mathcal{Y}$ (usually exponential in the input length). Consider the SVM formulation for structured output, which is a direct extension of the multi-class case:

Optimization Problem 2.7. (STRUCTURAL SUPPORT VECTOR MACHINES (MARGIN-RESCALING))

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ &s.t.\ \forall i, \forall \bar{y} \in \mathcal{Y}, \bar{y} \neq y_i \\ &\quad \boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, \bar{y}) \geq \Delta(y_i, \bar{y}) - \xi_i \\ &\quad \forall i, \xi_i \geq 0 \end{aligned} \tag{2.9}$$

The constraint essentially says that the score for the correct output $\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i)$ has to be greater than the score $\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y})$ for all alternative outputs $\bar{y}$. The loss function $\Delta(y_i, \bar{y})$ is new here. It controls the required margin between the two scores, and in the case of multi-class classification it is 0 if $y_i = \bar{y}$, and 1 otherwise. The larger the value of the loss, the more different y_i and $\bar{y}$ are, and the larger the required margin between them.

This optimization problem is still convex (convex objective with linear constraints). While on the surface the training problem and the semantics of the constraints are quite similar to the multi-class case, the exponential size of the output space $\mathcal{Y}$ makes a huge difference. For a start the constraint sets in structural SVMs, unlike in the multi-class case, cannot be explicitly enumerated.

Let us now go back to the POS tagging example we have earlier in the beginning of the chapter. We can map the input "I like support vector machines" and output "Noun Verb Noun Noun Noun" to a high dimensional feature space with a joint feature map $\boldsymbol{\Phi}(x, y)$. This joint feature map $\boldsymbol{\Phi}$ can be constructed in many different ways. For example, one way is to have one parameter for each of the conditional probabilities in the conditional probability tables in Table 2.1 (see Figure 2.6). Notice how the the joint feature map $\boldsymbol{\Phi}$ decomposes into emis-

$$
\begin{aligned}
&\Phi((\text{I}, \text{Like}, \text{Support}, \text{Vector}, \text{Machines}), (\text{Noun}, \text{Verb}, \text{Noun}, \text{Noun}, \text{Noun})) \\
&= \phi_e(\text{I}, \text{Noun}) + \phi_e(\text{Like}, \text{Verb}) + \phi_e(\text{Support}, \text{Noun}) + \phi_e(\text{Vector}, \text{Noun}) + \phi(\text{Machines}, \text{Noun}) \\
&\quad \phi_t(\text{Noun}, \text{Verb}) + \phi_t(\text{Verb}, \text{Noun}) + \phi_t(\text{Noun}, \text{Noun}) + \phi_t(\text{Noun}, \text{Noun}) + \phi_t(\text{Noun}, \text{Noun})
\end{aligned}
$$

$$
= \begin{pmatrix} 0 \\ \vdots \\ 2 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \vdots \end{pmatrix}
$$

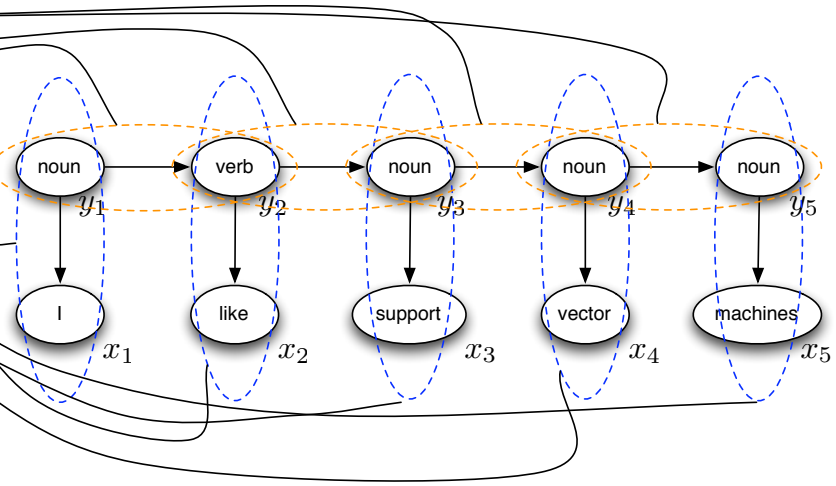


Figure 2.6: Illustration of the joint feature map $\boldsymbol{\Phi}$ applied to the HMM example

sion features $\boldsymbol{\phi}_e$ and transition features $\boldsymbol{\phi}_t$ in Figure 2.6. In this way the score $\boldsymbol{w} \cdot \boldsymbol{\Phi}(x, y)$ will be like the logarithm of the joint probability in Equation (2.1). But unlike the generative HMM case these parameters are not to be directly interpreted as conditional probabilities, and they are also much less constrained. For example, the parameters corresponding to transitions and emissions do not have to sum to one for each hidden state. This approach is sometimes referred to as "training a generative model discriminatively" in the literature [30].

Figure 2.7 shows the visualization of what the constraints in the structural SVM formulation mean for our POS tagging example. The loss function Δ is the Hamming loss here. We want to find a direction or weight vector $\boldsymbol{w}$ that ranks the correct output "Noun Verb Noun Noun Noun" the highest in the joint

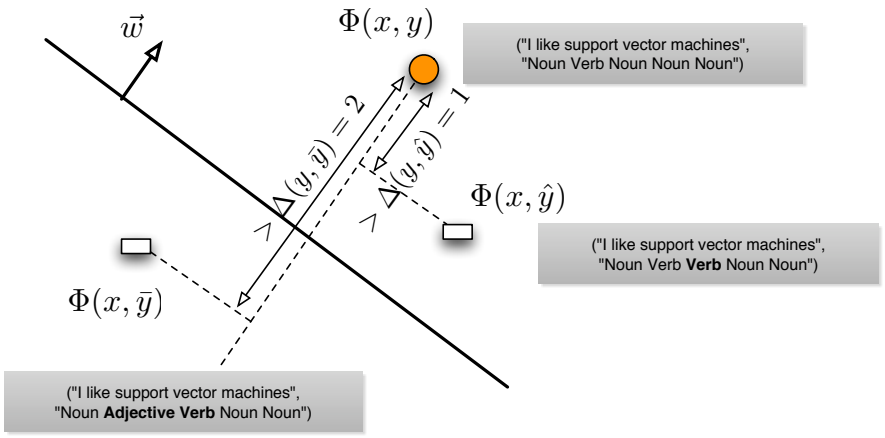


Figure 2.7: Structural Support Vector Machine on POS tagging example

feature space. The alternative output ”Noun Verb Verb Noun Noun” mislabels ”support” as a verb and has a Hamming loss of 1, so the constraint requires it to be at least distance 1 (in terms of the linear score $\boldsymbol{w} \cdot \boldsymbol{\Phi}(x, y)$) away from the correct output. Another output ”Noun Adj Verb Noun Noun” in addition mislabels ”like” as an adjective and has a Hamming loss of 2. So the constraint requires it to be at least distance 2 from the correct output.

The number of constraints in the Optimization Problem 2.7 is exponential in the length L, and indeed there are k^L ways to label a sequence of length L when there are k different parts-of-speech. Having a constraint set size that is exponential in the input instance size is very common in structured output prediction problems. Obviously this rules out the option of solving the optimization problem by enumerating all the constraints. The optimization problem can instead be solved by a cutting plane algorithm that generates constraints iteratively (which we will talk about in greater detail in the next chapter), and each iteration involves computing:

$$\text{argmax}_{\hat{y} \in \mathcal{Y}} [\boldsymbol{w} \cdot \boldsymbol{\Phi}(x, \hat{y}) + \Delta(y, \hat{y})]. \tag{2.10}$$

In our POS tagging example, as long as the joint feature vector and loss function decomposes linearly in hidden states y_i, then we can write the above problem as:

$$\text{argmax}_{\hat{y}\in\mathcal{Y}} \sum_{j=1}^{L} [\boldsymbol{w} \cdot \boldsymbol{\phi}_e(x, \hat{y}_j) + \boldsymbol{w} \cdot \boldsymbol{\phi}_t(\hat{y}_{j-1}, \hat{y}_j) + \delta(y_j, \hat{y}_j)], \tag{2.11}$$

where $\delta(y_j, \hat{y}_j) = 1$ if $y_j \neq \hat{y}_j$, and is 0 otherwise (since the Hamming loss is linear and $\Delta(y, \hat{y}) = \sum_{j=1}^{L} \delta(y_j, \hat{y}_j)$).

Notice that the structure of this argmax problem is exactly the same as the prediction problem we have in Equation (2.2), and thus can be solved by the Viterbi algorithm.

Moreover from Equation (2.11) we can see that the feature function $\boldsymbol{\phi}_e(x, \hat{y}_j)$ can have arbitrary dependence on the whole input sequence x, unlike in the generative HMM case where $\hat{y}_j$ is only allowed to depend on x_j due to the conditional independence assumptions. Thus we can construct much more descriptive features such as having $\hat{y}_j$ depend on x_{j-1} and x_{j+1} as well, or even a larger window of observations. Or we could construct other more global features such as the length of the input sequence. This extra flexibility of constructing arbitrary features from the input x is the major strength of discriminative modeling. Of course we also have to worry about overfitting when we introduce so many new features, but the regularization that comes with structural SVMs help control it.

2.4.1 Kernels in Structural SVMs

Like binary classification SVMs, structural SVMs can also be "kernelized" via the dual:

Optimization Problem 2.8. (STRUCTURAL SVM (DUAL))

$$\begin{aligned}
&\max_{\boldsymbol{\alpha}} \sum_{i,\hat{y}} \Delta(y_i, \hat{y}) \alpha_{i,\hat{y}} - \frac{1}{2} \sum_{i,\hat{y}} \sum_{j,\bar{y}} \alpha_{i,\hat{y}} \alpha_{j,\bar{y}} \boldsymbol{\Phi}(x_i, \hat{y}) \cdot \boldsymbol{\Phi}(x_j, \bar{y}) \\
&s.t. \sum_{\hat{y} \in \mathcal{Y}} \alpha_{i,\hat{y}} = C \\
&\quad \alpha_{i,\hat{y}} \geq 0 \quad \forall i, \forall \hat{y} \in \mathcal{Y}
\end{aligned}$$

The dual optimization only depends on the inner product of the joint feature vectors $\boldsymbol{\Phi}(x_i, \hat{y}) \cdot \boldsymbol{\Phi}(x_j, \bar{y})$, and thus can be replaced by a kernel function $K(x_i, \hat{y}, x_j, \bar{y})$. In practice usually not the whole joint feature vector is mapped nonlinearly to a high dimensional feature space, but only some constituient feature functions such as emissions are replaced with polynomial or Gaussian kernels. We will discuss kernels in structural SVMs in details in Chapter 6.

2.4.2 Slack-rescaling

Apart from the structural SVM formulation that we discuss above, there is another version that uses a different penalization method for the slack variables, which is called slack-rescaling in the literature.

Optimization Problem 2.9. (STRUCTUAL SVM (SLACK-RESCALING))

$$\begin{aligned}
&\min_{\boldsymbol{w}, \boldsymbol{\xi}} \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i \\
&s.t. \, \forall i, \forall \bar{y} \in \mathcal{Y}, \bar{y} \neq y_i \\
&\quad \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y}) \geq 1 - \frac{\xi_i}{\Delta(y_i, \bar{y})} \\
&\quad \forall i, \xi_i \geq 0
\end{aligned} \tag{2.12}$$

Here we assume that $\Delta(y_i, \bar{y}) > 0$ for $\bar{y} \neq y_i$, so that the constraints are well-defined. Compared to margin-rescaling we can see that the slack variable is penalized multiplicatively instead of additively by the loss. It would be even clearer if we re-organize the constraints:

$$\xi_i \geq \Delta(y_i, \hat{y})[1 - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y})] \quad \text{(slack rescaling)} \tag{2.13}$$

$$\xi_i \geq \Delta(y_i, \hat{y}) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y}) \quad \text{(margin rescaling)} \tag{2.14}$$

Slack-rescaling is less stringent than margin rescaling in the sense that as long as the score of the correct output $\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i)$ is greater than the score of any incorrect output $\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y})$ by 1 (or any fixed positive constant), then the slack ξ_i will be 0. Notice that satisfying this condition is sufficient for reproducing the correct output in the training set. On the other hand margin rescaling is more stringent in specifying the exact margin requirements. In practice slack-rescaling usually performs better on noisy structured output prediction problems, since unlike margin rescaling alternative outputs with scores with less than the correct output have little effect on the slack variable. However, slack rescaling is usually more expensive computationally during training due to non-additive nature of the loss [109]. In this thesis we will focus more on the margin-rescaling formulation because of its computational advantages.

2.5 Bibliographical Notes

2.5.1 Related Models

Below we give a brief review of different machine learning methods that are closely related to structural SVMs in the area of discriminative structured output learning.

Conditional Random Fields

The influential paper by Lafferty, McCallum and Pereira [81] on Conditional Random Fields (CRFs) is the work that opens up the whole area of discriminative learning of structured output prediction models. Conditional random fields learn the conditional probability distribution $P(y \mid x)$ with the following exponential form:

$$P(y \mid x) = \frac{\exp(\boldsymbol{w} \cdot \boldsymbol{\Phi}(x, y))}{\sum_{\hat{y} \in \mathcal{Y}} \exp(\boldsymbol{w} \cdot \boldsymbol{\Phi}(x, \hat{y}))}, \tag{2.15}$$

where $\boldsymbol{\Phi}$ is the joint feature vector that serves the same purpose as in structural SVMs, and $\boldsymbol{w}$ is the parameter vector to be learned. By modeling the conditional distribution $P(y \mid x)$ instead of the joint distribution $P(x, y)$, conditional random field is the first model that allows flexible feature construction in $\boldsymbol{\Phi}$, which greatly improves the performance of many sequence labeling tasks. Notice in Equation (2.15) the normalization factor only involves summation over outputs $\hat{y}$ but not input x, and therefore one can construct arbitrarily complex feature over the input x without having to worry about solving a more difficult inference problem in training. CRFs solve the following regularized negative log-likelihood minimization problem during training on a training set consist-

ing of $\{(x_1, y_1), \ldots, (x_n, y_n)\}$.

Optimization Problem 2.10. (CONDITIONAL RANDOM FIELDS (WITH REGULARIZATION))

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 - C\sum_{i=1}^{n}\left[\boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, y_i) - \log\left(\sum_{\hat{y}\in\mathcal{Y}}\exp(\boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, \hat{y}))\right)\right]. \tag{2.16}$$

The regularization term $1/2\|\boldsymbol{w}\|^2$ prevents the model from overfitting when the number of parameters is large. This training problem is a convex optimization problem and can be solved with methods such as iterative scaling [37] or limited memory BFGS [85, 113]. The form of this training problem is very similar to the training problem of structural SVMs in Equation (2.9), the only difference being the way we penalize the prediction errors on the training set. We can rewrite Equation (2.9) without the constraints for a direct comparison:

Optimization Problem 2.11. (STRUCTURAL SVM (AFTER ELIMINATING SLACK VARIABLES))

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\max_{\hat{y}\in\mathcal{Y}}[\boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, \hat{y}) - \boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, y_i) + \Delta(y_i, \hat{y})] \tag{2.17}$$

Compared to CRFs, structural SVMs contain a loss function Δ for specifying the margin requirements for different applications and do not require the computation of the normalization term $\sum_{\hat{y}\in\mathcal{Y}}\exp(\boldsymbol{w}\cdot\Phi(x, \hat{y}))$ (called the partition function) during training. For structured output prediction problems with inference procedures based on dynamic programming, the partition function can be computed using the sum-product algorithm (e.g., forward-backward algorithm in HMM), with computational complexity similar to the argmax computation (max-prodcut algorithm) used in the training of structural SVMs (e.g., Viterbi algorithm in HMM). For structured output prediction problems with

NP-hard inference problems, sometimes there are approximation algorithms for computing the highest scoring structure, which can be applied to the training of structural SVMs. However it is less clear how these algorithms can be modified to compute the partition function. Even for problems with polynomial time inference algorithms there could be differences in the computational complexity of computing the argmax and computing the partition function. For example, computing the minimum/maximum spanning tree takes $O(n)$ time with Kruskal or Prim's algorithm, but summing up all the scores of all possible spanning trees takes $O(n^3)$ time via the matrix-tree theorem [79]. However CRFs have the advantage of being a probabilistic model and can be composed with other probabilistic models in certain applications. It is also very well-grounded theoretically and can be derived from the principle of maximum entropy [5].

Margin-based Models

Collins [30] proposed using perceptron updates to learn the parameters of hidden Markov models, which allows the use of flexible feature functions like that in CRF, but at the same time is much easier to implement. Like moving from perceptrons to SVM for a more stable classification boundary, his work was extended in [6] to Hidden Markov Support Vector Machines (HM-SVM) using ideas of regularization. Compared with the perceptron training method, HM-SVM has improved accuracies on many sequence labeling tasks [6] due to regularization. HM-SVM was later generalized to structural SVMs [132] for general structured output learning.

Around the same time Taskar et al. took the idea from Collins one step further and apply it to the training of general Markov Random Fields, which they

called Max-Margin Markov Networks (M3N)[125]. The formulation of max-margin Markov networks is equivalent to the margin-rescaling loss penalization in structural SVMs. One major difference is their proposed training methods. M3N uses dual methods based on sequential minimal optimization (SMO) [102] while structural SVMs employ cutting plane algorithms for training. However these training methods are still slow on large datasets due to the repeated use of inference algorithms such as Viterbi decoding to compute gradients.

Trying to strike a balance between performance and training time, there were also works on introducing the concept of margins into online learning of structured output models, notably in [91]. It extends the MIRA algorithm for online learning [36] and have improved performance on dependency parsing over the Collins perceptron algorithm.

Kernel Dependency Estimation

Kernel Dependency Estimation (KDE) [144] takes a completely different approach to structured output prediction. The use of general kernels such as tree or string alignment kernels allows us to apply the SVM framework to classify input structures such as trees and sequences. KDE extends this idea by including a kernel on the output space as well, which maps the output structure to a high dimensional vector space. The learning task then becomes learning a mapping from the input kernel space to the output kernel space, which in their case was done using kernel PCA and regression. The major difficulty of this approach is that since the output kernel maps an output structure to the output kernel space, a preimage problem needs to be solved when making prediction. The advantage of this approach is that relatively simple methods such as kernel

logistic regression can be used for learning the mapping from the input kernel space to the output kernel space, without the need to perform a large number of Viterbi decoding as in the case of CRF or large-margin structured learning.

Search-based Structured Output Learning

Another distinct approach to structured-output learning is to relate it to another main area of machine learning, namely reinforcement learning. In [38] Daumé et al. relate the search process during the decoding in different structured output learning problem (Viterbi decoding, parsing, etc) to exploration in a state space. They apply ideas from reinforcement learning to perform structured output learning, a method which they call SEARN. They try to learn a cost-sensitive classifier to decide what action to take at a particular state in the search process based on past decisions (for example, what tag to use at the $(k+1)$th position after the first k tags are given in a left-to-right Viterbi decoding process). Compared to models such as CRFs or structural SVMs which perform discriminative parameter learning on graphical models [70], they try to model the search process directly and learn good parameters for the search function (policy) so that it will terminate at outputs with low loss. The experimental results are competitive with models like CRF and structural SVM on many natural language processing tasks.

2.5.2 Applications of Structural SVMs

Structural SVMs and related max-margin methods have been applied to a diverse set of structured output prediction problems in different fields. For exam-

ple, they have been applied to sequence tagging problems [6], parsing of natural language [126, 132], and noun phrase coreference resolution [47]. In computational biology they have been applied to the problem of protein sequence alignment [153] and prediction of secondary structures of proteins [50]. In information retrieval they have been applied to optimizing specific performance measures such as F1-scores and ROC-Area [65], Mean Average Precision [154], and topic coverage [155]. They have also been applied to hierarchical classification of taxonomies [20]. In computer vision there is work on applying structural SVMs to learn stereo vision [84], and in music there is work on chord transcription of music audio [143]. In most of these tasks state-of-art performance was obtained through the incorporation of extra features via the discriminative modeling and large-margin robustness of structural SVMs.

CHAPTER 3

CUTTING PLANE ALGORITHMS FOR TRAINING STRUCTURAL SVMS

In the last chapter we have introduced structural SVM, a discriminative framework for learning structured output prediction problems. We have seen how the discriminative modeling power of structural SVMs allow the introduction of flexible features into structured output learning problems and their numerous applications. However, all these benefits cannot materialize if we cannot solve the training problem of structural SVM efficiently. In this chapter we are going to discuss an efficient method of solving structural SVMs based on the cutting plane algorithm.

Consider again Optimization Problem 2.7, which is a convex quadratic program with linear constraints.

Optimization Problem.

$$\begin{aligned}
&\min_{\boldsymbol{w},\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\
&s.t.\ \forall i, \forall \bar{y} \in \mathcal{Y}, \bar{y} \neq y_i \\
&\qquad \boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, \bar{y}) \geq \Delta(y_i, \bar{y}) - \xi_i \\
&\qquad \forall i, \xi_i \geq 0
\end{aligned}$$

The major difficulty with solving this QP is the large number of constraints: for example in POS-tagging with k types of parts-of-speech, on a sequence of length L, there are k^L many possible tag sequences $\bar{y}$. Obviously this rules out the possibility of enumerating all the constraints on $\bar{y} \neq y_i$ and solve the QP using a conventional QP solver. However, perhaps a bit surprisingly, not all constraints are needed when we solve for the optimal $\boldsymbol{w}$ in Optimization Prob-

lem 2.7. We can use a cutting plane algorithm [75] to generate the constraints one by one and solve the QP over this restricted constraint set. The surprising fact is that for a fixed precision ϵ the number of constraints added by the cutting plane algorithm is polynomial in the size of each training example (e.g. length of sequences in POS-tagging, or length of protein sequence pairs to be aligned) and the total number of examples in the training set. This approach is very similar to the use of column generation in the solution of linear programs.

3.1 Cutting Plane Algorithm

Before discussing the cutting plane algorithm, let us first rewrite Optimziation Problem 2.7 into a form that is simpler to describe the algorithm in. First of all notice that Optimization Problem 2.7 is equivalent to the following problem:

Optimization Problem 3.1. (STRUCTURAL SVM (n-SLACK FORMULATION))

$$\begin{aligned} &\min_{\boldsymbol{w},\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\ &s.t.\ \forall i, \forall \bar{y} \in \mathcal{Y}, \\ &\qquad \boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, \bar{y}) \geq \Delta(y_i, \bar{y}) - \xi_i \end{aligned} \tag{3.1}$$

if we assume $\Delta(y_i, y_i) = 0$ for all labels y_i in the training set (which is a condition satisfied by most loss functions) and rescaling C by a factor of n. We divide C by n because we want to consider the average loss or average slack of the training sample. Notice the set of constraints extends from all $\bar{y} \neq y_i$ to all possible outputs $\bar{y} \in \mathcal{Y}$, and the non-negative constraints for the slack variables have been eliminated. The two optimization problems are equivalent because if

we substitute $\bar{y} = y_i$ in Optimization Problem 2.7, we obtain

$$\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) \geq \Delta(y_i, y_i) - \xi_i,$$

which is equivalent to $\xi_i \geq 0$.

The next rewrite transforms Optimization Problem 3.1 into one that only involves one slack variable:

Optimization Problem 3.2. (STRUCTURAL SVM (1-SLACK FORMULATION))

$$\begin{aligned} &\min_{\boldsymbol{w},\xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\xi \\ &s.t.\ \forall(\bar{y}_1, \ldots, \bar{y}_n) \in \mathcal{Y}^n, \\ &\quad \frac{1}{n}\sum_{i=1}^{n}[\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y}_i)] \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i, \bar{y}_i) - \xi \end{aligned} \tag{3.2}$$

The equivalence can be observed quite easily if we rewrite Optimization Problem 3.2 in a form that eliminates all slack variables:

$$\begin{aligned} &\frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\max_{\bar{y}\in\mathcal{Y}}[\Delta(y_i, \bar{y}) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y})] \\ = &\frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{n}\max_{(\bar{y}_1,\ldots,\bar{y}_n)\in\mathcal{Y}^n}\sum_{i=1}^{n}[\Delta(y_i, \bar{y}_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y}_i)]. \end{aligned}$$

The max over the product space $\mathcal{Y}^n$ in the second line distributes over the independent summands $[\Delta(y_i, \bar{y}_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y}_i)]$ and is therefore equivalent to the first line.

Although at a first glance Optimization Problem 3.2 looks even more complicated than Optimization Problem 3.1 as the constraints now range over a larger product space, it is actually simpler as we only need to generate constraints for one slack variable. We will show shortly that the number of constraints added will be polynomial in the input size. For now let us first look at the cutting plane algorithm in Algorithm 3.1 for solving Optimization Problem 3.2:

1: Input: $S = ((x_1, y_1), \ldots, (x_n, y_n))$, C, ϵ
2: $\mathcal{W} \leftarrow \emptyset$
3: **repeat**
4: $(\boldsymbol{w}, \xi) \leftarrow \text{argmin}_{\boldsymbol{w},\xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\xi$
s.t. $\forall(\bar{y}_1, \ldots, \bar{y}_n) \in \mathcal{W} : \xi \geq \frac{1}{n}\sum_{i=1}^{n}[\Delta(y_i, \bar{y}_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y}_i)]$
5: **for** $i = 1, \ldots, n$ **do**
6: $\bar{y}_i \leftarrow \text{argmax}_{\bar{y} \in \mathcal{Y}}[\Delta(y_i, \bar{y}) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y})]$
7: **end for**
8: $\mathcal{W} \leftarrow \mathcal{W} \cup \{(\bar{y}_1, \ldots, \bar{y}_n)\}$
9: **until** $\frac{1}{n}\sum_{i=1}^{n}[\Delta(y_i, \bar{y}_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y}_i)] \leq \xi + \epsilon$
10: return $(\boldsymbol{w}, \xi)$

Algorithm 3.1: Cutting Plane Algorithm for training Structural SVMs (Primal)

The basic idea of the cutting plane algorithm is quite simple: as long as there is a constraint that violates the current solution $(\boldsymbol{w}, \xi)$ by more than ϵ, i.e., $\frac{1}{n}\sum_{i=1}^{n}[\Delta(y_i, \bar{y}_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y}_i)] > \xi + \epsilon$, add that constraint to the current working set $\mathcal{W}$ and resolve the QP. Figure 3.1 illustrate the successive steps in a cutting plane algorithm using a simple 1-dimensional optimization problem. Step 0 shows the original objective and the circle shows our starting point. At step 1 we generate the first cutting plane, which is a linear under-approximation to the objective function. We minimize over this linear approximation and move our current solution to the right, indicated by the circle (we omit the regularizer and assume the domain of optimization is bounded in this simple illustration). In step 2 we generate a second cutting plane, and minimize over the new cutting plane model, which is the maximum over the two cutting planes. Then we keep on generating new cutting planes to improve our model and re-optimize, as illustrated in steps 3 and 4. We stop the cutting plane algorithm when the difference between the true objective and the cutting plane model is small, say, less than ϵ (step 4).

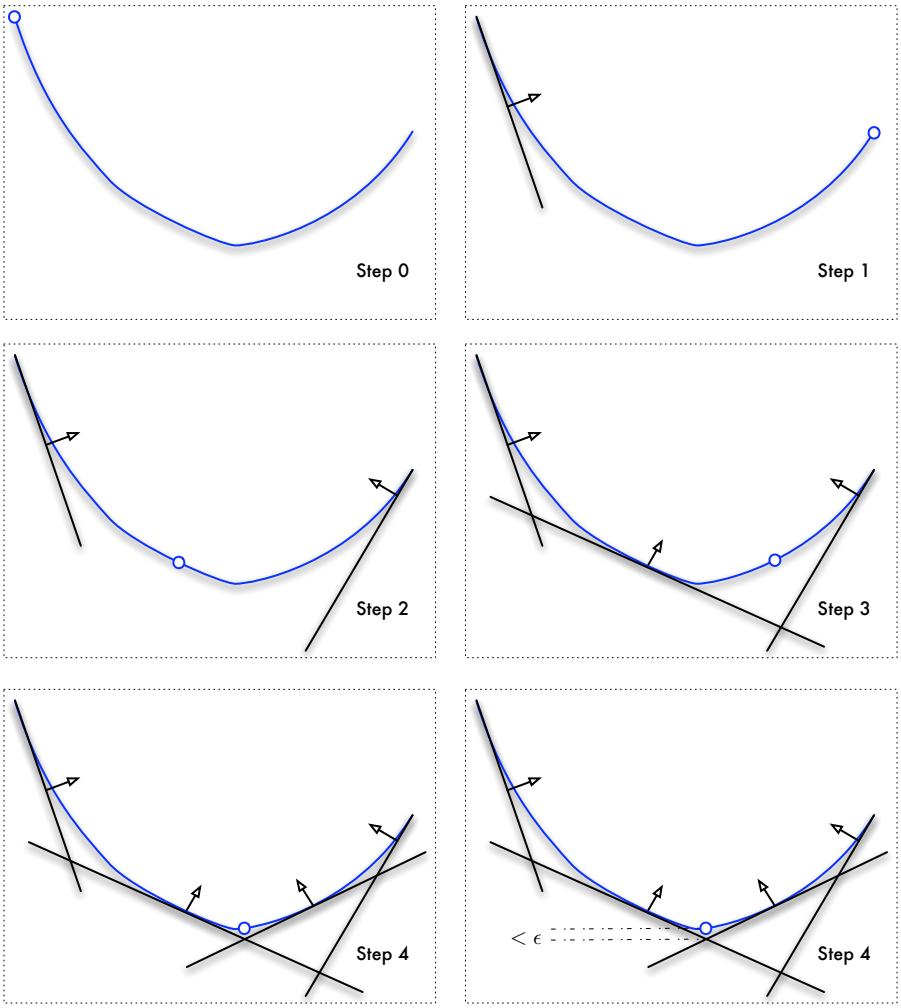


Figure 3.1: Illustration of cutting plane algorithm: the blue curve is the true objective function, while the black straight lines are the cutting planes.

It is clear that at termination (assuming it always terminates) there will be no more ϵ-violated constraints. A fairly easy consequence is that the solution returned $(\boldsymbol{w}_t, \xi_t)$ will have an objective o_t no more than $C\epsilon$ from the objective o^*

of the optimal solution $(\boldsymbol{w}^*, \xi^*)$.

$$
\begin{aligned}
& o^* \\
&= \frac{1}{2}\|\boldsymbol{w}^*\|^2 + C \max_{(\bar{y}_1,\ldots,\bar{y}_n)\in\mathcal{Y}^n} \frac{1}{n}\sum_{i=1}^{n}[\Delta(y_i,\bar{y}_i) - \boldsymbol{w}^*\cdot\boldsymbol{\Phi}(x_i,y_i) + \boldsymbol{w}^*\cdot\boldsymbol{\Phi}(x_i,\bar{y}_i)] \\
&\geq \frac{1}{2}\|\boldsymbol{w}^*\|^2 + C \max_{(\bar{y}_1,\ldots,\bar{y}_n)\in\mathcal{W}} \frac{1}{n}\sum_{i=1}^{n}[\Delta(y_i,\bar{y}_i) - \boldsymbol{w}^*\cdot\boldsymbol{\Phi}(x_i,y_i) + \boldsymbol{w}^*\cdot\boldsymbol{\Phi}(x_i,\bar{y}_i)] \\
&\geq \frac{1}{2}\|\boldsymbol{w}_t\|^2 + C \max_{(\bar{y}_1,\ldots,\bar{y}_n)\in\mathcal{W}} \frac{1}{n}\sum_{i=1}^{n}[\Delta(y_i,\bar{y}_i) - \boldsymbol{w}_t\cdot\boldsymbol{\Phi}(x_i,y_i) + \boldsymbol{w}_t\cdot\boldsymbol{\Phi}(x_i,\bar{y}_i)] \\
&\geq \frac{1}{2}\|\boldsymbol{w}_t\|^2 + C \left(\max_{(\bar{y}_1,\ldots,\bar{y}_n)\in\mathcal{Y}^n} \frac{1}{n}\sum_{i=1}^{n}[\Delta(y_i,\bar{y}_i) - \boldsymbol{w}_t\cdot\boldsymbol{\Phi}(x_i,y_i) + \boldsymbol{w}_t\cdot\boldsymbol{\Phi}(x_i,\bar{y}_i)] - \epsilon \right) \\
&= o_t - C\epsilon
\end{aligned}
$$

Thus $o_t - o^* \leq C\epsilon$.

In practice we usually solve the inner quadratic optimization problem over the working set $\mathcal{W}$ via the dual, and the dual algorithm is presented in Algorithm 3.2. We will show how the dual is derived in the proof of the iteration bound below.

What is less clear is whether this algorithm actually terminates and how many cutting planes it would need to add before termination. The following theorem settles this question:

Theorem 3.1. *For any $C > 0$, $0 < \epsilon \leq 4R^2C$ and any training sample $S = ((x_1,y_1),\ldots,(x_n,y_n))$, Algorithm 3.1 terminates after adding at most*

$$\left\lceil \log_2\left(\frac{\bar{\Delta}}{4R^2C}\right)\right\rceil + \left\lceil \frac{16R^2C}{\epsilon}\right\rceil$$

cutting planes. $R^2 = \max_{i,\bar{y}} \|\boldsymbol{\Phi}(x_i,y_i) - \boldsymbol{\Phi}(x_i,\bar{y})\|^2$, $\bar{\Delta} = \max_{i,\bar{y}} \Delta(y_i,\bar{y})$, and $\lceil .. \rceil$ is the integer ceiling function.

Before we give the full proof of the above results, we give an outline of the main proof idea. This iteration bound is constructed via a dual lower bound

1: Input: $S = ((x_1, y_1), \ldots, (x_n, y_n))$, C, ϵ
2: $\boldsymbol{c} \leftarrow \mathbf{0}$, $\boldsymbol{H} \leftarrow \mathbf{0}$, $t \leftarrow 0$
3: **repeat**
4: $\quad \boldsymbol{H} \leftarrow (\boldsymbol{H}_{ij})_{1 \le i,j \le t}$, where $\boldsymbol{H}_{ij} = \boldsymbol{g}^{(i)} \cdot \boldsymbol{g}^{(j)}$
5: $\quad \boldsymbol{\alpha} \leftarrow \text{argmax}_{\boldsymbol{\alpha} \ge 0} \boldsymbol{\alpha}^T \boldsymbol{c} - \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha}$ s.t. $\boldsymbol{\alpha}^T \mathbf{1} \le C$
6: $\quad \xi \leftarrow \frac{1}{C}(\boldsymbol{\alpha}^T \boldsymbol{c} - \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha})$
7: $\quad \boldsymbol{w} \leftarrow \sum_{i=1}^{t} \alpha_i \boldsymbol{g}^{(t)}$
8: $\quad t \leftarrow t + 1$
9: $\quad$ **for** i=1,...,n **do**
10: $\quad\quad \bar{y}_i \leftarrow \text{argmax}_{\bar{y} \in \mathcal{Y}} [\Delta(y_i, \bar{y}) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y})]$
11: $\quad$ **end for**
12: $\quad \boldsymbol{g}^{(t)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} [\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \bar{y}_i)]$
13: $\quad c^{(t)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \Delta(y_i, \bar{y}_i)$
14: **until** $\boldsymbol{w} \cdot \boldsymbol{g}^{(t)} \ge c^{(t)} - \xi - \epsilon$

Algorithm 3.2: Cutting Plane Algorithm for training Structural SVMs (Dual)

argument. The feasible solution of the dual of the Optimization Problem 3.2 is a lower bound of the optimal value. We can show that every time we add an ϵ-violated constraint the dual increases by at least an amount $\delta(\epsilon)$ (a function of ϵ). Since the $\boldsymbol{w} = 0, \xi = \bar{\Delta}$ is a feasible primal solution with value $C\bar{\Delta}$, the maximum dual value must be no more than $C\bar{\Delta}$. Therefore we can add no more than $C\bar{\Delta}/\delta(\epsilon)$ constraints. This argument gives rise to the $O(C/\epsilon^2)$ bound in [132]. Below we reproduce the proof given in [67], which provides a tighter $O(C/\epsilon)$ bound through more careful analysis. This bound will also be useful when we discuss applying the cutting plane algorithm with approximation for training structural SVMs with nonlinear kernels.

3.2 Proof of Iteration Bound

First of all let us derive the dual of Optimization Problem 3.2 via the Lagrangian. We first introduce one dual variable $\alpha_{\bar{\boldsymbol{y}}}$ for each constraint, where $\bar{\boldsymbol{y}} = (\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n)$ are the most violated output structure in the argmax computation at Line 6 of Algorithm 3.1. We have the following Lagrangian:

$$L(\boldsymbol{w}, \xi, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\xi + \sum_{\bar{\boldsymbol{y}} \in \mathcal{Y}^n} \alpha_{\bar{\boldsymbol{y}}} \left(\frac{1}{n} \sum_{i=1}^{n} [\Delta(y_i, \bar{y}_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y}_i)] - \xi \right)$$

Differentiating with respect to $\boldsymbol{w}$ and setting the derivatives to zero gives:

$$\boldsymbol{w} = \sum_{\bar{\boldsymbol{y}} \in \mathcal{Y}^n} \alpha_{\bar{\boldsymbol{y}}} \frac{1}{n} \sum_{i=1}^{n} [\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \bar{y}_i)]$$

Similarly differentiating with respect to ξ gives:

$$\sum_{\bar{\boldsymbol{y}} \in \mathcal{Y}^n} \alpha_{\bar{\boldsymbol{y}}} = C$$

Plugging $\boldsymbol{w}$ back into the Lagrangian and with the constraints on $\boldsymbol{\alpha}$ we have the following dual optimization problem:

$$\begin{aligned}
\max_{\boldsymbol{\alpha}} & \sum_{\bar{\boldsymbol{y}} \in \mathcal{Y}^n} \alpha_{\bar{\boldsymbol{y}}} \left(\frac{1}{n} \sum_{i=1}^{n} \Delta(y_i, \bar{y}_i) \right) \\
& - \frac{1}{2} \sum_{\bar{\boldsymbol{y}} \in \mathcal{Y}^n} \sum_{\bar{\boldsymbol{y}}' \in \mathcal{Y}^n} \alpha_{\bar{\boldsymbol{y}}} \alpha_{\bar{\boldsymbol{y}}'} \left(\frac{1}{n} \sum_{i=1}^{n} [\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \bar{y}_i)] \right) \cdot \left(\frac{1}{n} \sum_{j=1}^{n} [\boldsymbol{\Phi}(x_j, y_j) - \boldsymbol{\Phi}(x_j, \bar{y}'_j)] \right) \\
& \sum_{\bar{\boldsymbol{y}} \in \mathcal{Y}^n} \alpha_{\bar{\boldsymbol{y}}} = C \\
& \forall \bar{\boldsymbol{y}} \in \mathcal{Y}^n, \alpha_{\bar{\boldsymbol{y}}} \geq 0
\end{aligned}$$

Notice that this is the same as the dual optimization problem in Algorithm 3.2, except the optimization problem in Algorithm 3.2 is restricted to dual variables $\alpha_{\bar{\boldsymbol{y}}}$ in the working set $\bar{\boldsymbol{y}} \in \mathcal{W}$ only.

We shall now lower bound the increase in the dual objective of Optimization Problem 3.2. For this we need a lemma:

Lemma 3.1. *Consider the unconstrained quadratic program,*

$$\max_{\boldsymbol{\alpha}} D(\boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}} \boldsymbol{h}^T\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{H}\boldsymbol{\alpha}$$

with positive semidefinite $\boldsymbol{H}$. Any line search starting from $\boldsymbol{\alpha}$ in the direction $\boldsymbol{\eta}$ with maximum step size C will increase the objective by at least

$$\max_{0\leq\beta\leq C} D(\boldsymbol{\alpha}+\beta\boldsymbol{\eta}) - D(\boldsymbol{\alpha}) \geq \frac{1}{2}\min\{C, \frac{\nabla D(\boldsymbol{\alpha})^T\boldsymbol{\eta}}{\boldsymbol{\eta}^T\boldsymbol{H}\boldsymbol{\eta}}\}\nabla D(\boldsymbol{\alpha})^T\boldsymbol{\eta}$$

Proof. Writing down the line search objective in terms of β we have

$$\begin{aligned} & D(\boldsymbol{\alpha}+\beta\boldsymbol{\eta}) - D(\boldsymbol{\alpha}) \\ &= \beta\boldsymbol{h}^T\boldsymbol{\eta} - \frac{1}{2}\beta^2\boldsymbol{\eta}^T\boldsymbol{H}\boldsymbol{\eta} - \beta\boldsymbol{\alpha}^T\boldsymbol{H}\boldsymbol{\eta} \end{aligned} \tag{3.3}$$

The unconstrained maximizer of this quadratic function of β is obtained at

$$\beta = \frac{\boldsymbol{h}^T\boldsymbol{\eta} - \boldsymbol{\alpha}^T\boldsymbol{H}\boldsymbol{\eta}}{\boldsymbol{\eta}^T\boldsymbol{H}\boldsymbol{\eta}} = \frac{\nabla D(\boldsymbol{\alpha})^T\boldsymbol{\eta}}{\boldsymbol{\eta}^T\boldsymbol{H}\boldsymbol{\eta}}$$

if $\boldsymbol{\eta}^T\boldsymbol{H}\boldsymbol{\eta} > 0$.

Since $\boldsymbol{\eta}$ is an ascent direction, the maximum is either attained at the above value of β for the unconstrained maximum or at the maximum step size C (since there is only 1 turning point in a quadratic curve). The step size C also covers the case for $\boldsymbol{\eta}^T\boldsymbol{H}\boldsymbol{\eta} = 0$.

Plugging back these two values of β ($\nabla D(\boldsymbol{\alpha})^T\boldsymbol{\eta}/\boldsymbol{\eta}^T\boldsymbol{H}\boldsymbol{\eta}$ and C) into Equation (3.3), we obtain the desired bound. □

We implicity assumed that all the entries in the dual variable vector $\boldsymbol{\alpha}$ are zero for any $\bar{\boldsymbol{y}}$ not in the constraint set $\mathcal{W}$. Let the dual solution at the ith iteration be $\boldsymbol{\alpha}^{(i)}$, so that the current solution (iteration t) is $\boldsymbol{\alpha}^{(t)}$. When a new

ϵ-violated constraint $\hat{\boldsymbol{y}}$ is added, we shall consider the line search direction defined by:

$$\boldsymbol{\eta} = \boldsymbol{e}_{\hat{\boldsymbol{y}}} - \frac{1}{C}\boldsymbol{\alpha}^{(t)},$$

where $\boldsymbol{e}_{\hat{\boldsymbol{y}}}$ is the vector with 1 at entry for $\hat{\boldsymbol{y}}$ and 0 elsewhere. We want to lower bound the increase in the dual objective when a line search is performed:

$$\max_{0 \leq \beta \leq C} [D(\boldsymbol{\alpha}^{(t)} + \beta\boldsymbol{\eta}) - D(\boldsymbol{\alpha}^{(t)})]$$

Notice that this search direction maintains the equality constraint $\sum_{\bar{\boldsymbol{y}} \in \mathcal{Y}^n} \alpha_{\bar{\boldsymbol{y}}} = C$ for any positive step size β, and restricting $\beta \leq C$ ensures that the non-negativity constraint on $\boldsymbol{\alpha}$ are satisfied. Also note that although we are not using this line search directly in Algorithm 3.2, solving the QP in Line 5 directly over all the cutting planes generated is guaranteed to return a solution $\boldsymbol{\alpha}^{(t+1)}$ with higher dual objective $D(\boldsymbol{\alpha}^{(t+1)})$ than the one returned by the above line search direction.

To apply Lemma 3.1 we need to bound $\nabla D(\boldsymbol{\alpha})^T \boldsymbol{\eta}$.

$$\begin{aligned}
\frac{\partial D(\alpha)}{\partial \alpha_{\bar{\boldsymbol{y}}}} &= \frac{1}{n}\sum_{i=1}^{n} \Delta(y_i, \bar{y}_i) \\
&\quad - \left(\frac{1}{n}\sum_{i=1}^{n} [\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \bar{y}_i)]\right) \cdot \sum_{\bar{\boldsymbol{y}}' \in \mathcal{Y}^n} \alpha_{\bar{\boldsymbol{y}}'} \left(\frac{1}{n}\sum_{i=1}^{n} [\boldsymbol{\Phi}(x_j, y_j) - \boldsymbol{\Phi}(x_j, \bar{y}'_j)]\right) \\
&= \frac{1}{n}\sum_{i=1}^{n} \Delta(y_i, \bar{y}_i) - \left(\frac{1}{n}\sum_{i=1}^{n} [\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \bar{y}_i)]\right) \cdot \boldsymbol{w} \\
&= \xi
\end{aligned}$$

for all those constraints with $\alpha_{\bar{\boldsymbol{y}}} > 0$ (by complimentary slackness condition).

And for the new constraint $\hat{\boldsymbol{y}}$ added we have:

$$\begin{aligned}
\frac{\partial D(\alpha)}{\partial \alpha_{\hat{\boldsymbol{y}}}} &= \frac{1}{n}\sum_{i=1}^{n} \Delta(y_i, \hat{y}_i) - \left(\frac{1}{n}\sum_{i=1}^{n} [\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \hat{y}_i)]\right) \cdot \boldsymbol{w} \\
&= \xi + \gamma
\end{aligned}$$

for some $\gamma > \epsilon$, since the stopping criterion is not satisfied when a new constraint is added.

Therefore

$$\begin{aligned}
&\nabla D(\boldsymbol{\alpha}_t)^T \boldsymbol{\eta} \\
&= (\xi + \gamma) - \frac{1}{C}(\xi \sum_{\bar{\boldsymbol{y}} \in \mathcal{Y}^n} \alpha_{\bar{\boldsymbol{y}}}) \\
&= \gamma.
\end{aligned}$$

Moreover

$$\begin{aligned}
&\boldsymbol{\eta}^T \boldsymbol{H} \boldsymbol{\eta} \\
&= \|\boldsymbol{g}^{(t+1)} - \frac{1}{C} \sum_{j=1}^{t} \alpha_{\bar{\boldsymbol{y}}_j} \boldsymbol{g}^{(j)}\|^2 \\
&\leq (\|\boldsymbol{g}^{(t+1)}\| + \frac{1}{C} \sum_{j=1}^{t} \alpha_{\bar{\boldsymbol{y}}_j} \|\boldsymbol{g}^{(j)}\|)^2 \\
&= (R + \frac{1}{C} \sum_{j=1}^{t} \alpha_{\bar{\boldsymbol{y}}_j} R)^2 \\
&= 4R^2,
\end{aligned}$$

where $\boldsymbol{g}^{(j)}$ is shorthand for the cutting plane $\frac{1}{n} \sum_{i=1}^{n} [\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \bar{y}_i)]$ generated at the jth iteration in Algorithm 3.2.

Plugging back into the bounds of Lemma 3.1 we obtain

$$\max_{0 \leq \beta C} D(\boldsymbol{\alpha} + \beta \boldsymbol{\eta}) - D(\boldsymbol{\alpha}) \geq \min\{\frac{C\gamma}{2}, \frac{\gamma^2}{8R^2}\} \tag{3.4}$$

Let $\boldsymbol{\alpha}^*$ be the optimal solution. Define the optimality gap of the ith iteration by $\delta(i) = D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^{(i)})$. Let $\boldsymbol{y^*}$ be the vector of correct labels $\boldsymbol{y^*} = (y_1, y_2, \ldots, y_n)$. Suppose we start from the initial solution $\boldsymbol{\alpha}^{(0)}$ where $\alpha^{(0)}_{\boldsymbol{y^*}} = C$ and $\alpha^{(0)}_{\bar{\boldsymbol{y}}} = 0$ for all other $\bar{\boldsymbol{y}} \in \mathcal{Y}^n$. Notice $\boldsymbol{\alpha}^{(0)}$ is a dual feasible solution with dual

objective 0. As $\boldsymbol{w} = 0, \xi = \bar{\Delta}$ is a primal feasible solution with primal objective $C\bar{\Delta}$, we have initial optimality gap $\delta(0) = D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^{(0)}) \leq C\bar{\Delta} - 0 = C\bar{\Delta}$.

Let γ_i be the violation at the ith iteration, and let $P(\boldsymbol{\alpha})$ be the primal objective of Optimization Problem 3.2 by putting $\boldsymbol{w} = \sum_{j=1}^{i} \alpha_{\bar{\boldsymbol{y}}_j} \boldsymbol{g}^{(j)}$, we have by weak duality:

$$\begin{aligned} \gamma_i &= \frac{1}{C}[P(\boldsymbol{\alpha}^{(i)}) - D(\boldsymbol{\alpha}^{(i)})] \\ &\geq \frac{1}{C}[D(\boldsymbol{\alpha}^*) - D(\boldsymbol{\alpha}^{(i)})] \\ &= \frac{\delta(i)}{C} \end{aligned} \tag{3.5}$$

Combining Equations (3.4) and (3.5) we have:

$$\begin{aligned} & D(\boldsymbol{\alpha}^{(i+1)}) - D(\boldsymbol{\alpha}^{(i)}) \\ \geq\ & \min\{\frac{C\gamma_i}{2}, \frac{\gamma_i^2}{8R^2}\} \\ \geq\ & \min\{\frac{\delta(i)}{2}, \frac{\delta(i)^2}{8R^2C^2}\} \end{aligned}$$

Note that $D(\boldsymbol{\alpha}^{(i+1)}) - D(\boldsymbol{\alpha}^{(i)}) = \delta(i) - \delta(i+1)$, and thus we have the recurrence:

$$\delta(i) - \delta(i+1) \geq \min\{\frac{\delta(i)}{2}, \frac{\delta(i)^2}{8R^2C^2}\} \tag{3.6}$$

The first case occurs when $\delta(i) \geq 4R^2C^2$, and by solving the recurrence

$$\begin{aligned} \delta(i) - \delta(i+1) &\geq \frac{\delta(i)}{2} \\ \delta(i+1) &\leq \frac{\delta(i)}{2} \end{aligned}$$

with $\delta(0) \leq C\bar{\Delta}$, we conclude that it takes at most

$$i_1 = \lceil \log_2 \frac{\bar{\Delta}}{4R^2C} \rceil$$

iterations to reduce the optimality gap to below $4R^2C^2$.

After that the second case of the reccurence in Equation (3.6) holds, and we have:

$$\delta(i) - \delta(i+1) \geq \frac{\delta(i)^2}{8R^2C^2}$$

for $i > i_1$. For this we can follow the approach in [128] and upper bound $\delta(i)$ with the differential equation

$$\frac{\partial \delta(i)}{\partial i} = \frac{1}{8R^2C^2}\delta(i)^2$$

with boundary condition $\delta(i_1) = 4R^2C^2$. The solution gives $\delta(i) \leq 8R^2C^2/(i+2)$. Hence it takes no more than

$$i_2 = \frac{8R^2C}{\epsilon}$$

iterations to reduce the optimality gap to below $C\epsilon$. After that it is no longer guaranteed that there are ϵ-violated constraints. However at this point each ϵ-constraint we add will increase the dual objective by at least $\epsilon^2/8R^2$. Hence we can at most add

$$i_3 = \lceil \frac{8R^2C}{\epsilon} \rceil$$

such ϵ-constraints. Adding i_1, i_2 and i_3 gives the desired iteration bound.

3.3 Loss-Augmented Inference

The basic cutting plane algorithm as illustrated above is quite simple: alternate between generating cutting planes and re-solving the quadratic program. However in discriminative structured output learning it is not uncommon to repeat the above process for several hundred iterations. Solving the dual QP above with several hundred variables is very fast (usually just a few seconds with interior point method solvers), but the generation of cutting plane in Line 10 of

Algorithm 3.2, i.e.,

$$\bar{y} \leftarrow \text{argmax}_{\bar{y} \in \mathcal{Y}}[\Delta(y, \bar{y}) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x, \bar{y})],$$

is usually the most expensive part. This is especially true when we have a large training set (n is large) or the underlying structure of the argmax problem is complex. This argmax computation is usually referred to as the *loss-augmented inference* problem in the literature, since the score is modified by a loss function when we try to find out an output structure that maximizes the score.

Let us go back to the POS-tagging example in Chapter 2. Suppose we use the Hamming loss in the learning, which is linearly decomposable in the individual tags y_i in the output sequence y (we use y_i to denote the ith state of the sequence y instead of the ith example in this part),

$$\Delta_{Hamming}(y, \bar{y}) = \sum_{i=1}^{L} \delta(y_i, \bar{y}_i),$$

where $\delta(y_i, \bar{y}_i) = 1$ if $y_i \neq \bar{y}_i$, and 0 otherwise.

The argmax computation in the POS-tagging case can be written as

$$\begin{aligned} &\underset{\bar{y} \in \mathcal{Y}}{\text{argmax}} \left[\sum_{i=1}^{L} \delta(y_i, \bar{y}_i) + \boldsymbol{w} \cdot \left(\boldsymbol{\phi}_e(x_1, \bar{y}_1) + \sum_{i=2}^{L} \boldsymbol{\phi}_e(x_i, \bar{y}_i) + \boldsymbol{\phi}_t(\bar{y}_{i-1}, \bar{y}_i) \right) \right] \\ = &\underset{\bar{y} \in \mathcal{Y}}{\text{argmax}} \left[\boldsymbol{w} \cdot \boldsymbol{\phi}_e(x_1, \bar{y}_1) + \delta(y_1, \bar{y}_1) + \sum_{i=2}^{L} (\boldsymbol{w} \cdot \boldsymbol{\phi}_e(x_i, \bar{y}_i) + \delta(y_i, \bar{y}_i) + \boldsymbol{w} \cdot \boldsymbol{\phi}_t(\bar{y}_{i-1}, \bar{y}_i)) \right]. \end{aligned}$$

This is of the same form as Equation (2.2)

$$\text{argmax}_{y \in \mathcal{Y}} P(x, y) = \text{argmax}_{y \in \mathcal{Y}} P(y_1) P(x_1 \mid y_1) \prod_{i=2}^{L} P(y_i \mid y_{i-1}) P(x_i \mid y_i),$$

and we can solve it using the same Viterbi algorithm.

So while the cutting plane generation can be computed via the familiar Viterbi algorithm, it is still expensive when we have to do it repeatedly over

a large training set. The cost of Viterbi algorithm is $O(k^2 L)$ for a first-order HMM with state space size k and sequence length L. Other structured output learning problems with dynamic programming based inference might have even higher computational complexity for solving the argmax problem, for example sequence alignments ($O(L^2)$ for sequence pairs of length L) and parsing in natural language processing ($O(L^3)$ for sentences of length L). There are even more difficult inference problems such as correlation clustering, decoding in machine translation [78] or labeling general Markov Random Fields with cycles, which are all NP-hard. For these problems approximation algorithms have to be employed in their argmax computations, and the use of approximation in conjunction with discriminative structured output learning has been investigated by several different groups [80, 48, 89]. Investigating the use of these inference algorithms in structural SVMs is not the focus of this thesis, but we would like to emphasize the importance of designing a joint feature map $\mathbf{\Phi}$ and loss function Δ that allows for efficient cutting plane generation when applying structural SVMs to different problems.

3.4 Bibliographical Notes

The cutting plane algorithm [75] and related bundle methods [77, 60] have been employed in covnex non-smooth optimization for a long time. However its use in training linear SVMs within the machine learning community is relatively recent. Chapelle [23] proposed training linear SVMs in the primal using a smooth version of the hinge loss, while Joachims [66] was to first to propose training linear SVMs with the cutting plane algorithm. The cutting plane algorithm for training structural SVMs was first published at [67], with the convergence proof

also first appearing in there. It built on work on applying the cutting plane algorithm to train binary classification SVMs [66], and also the improved theoretical analysis that first appears in [128]. The algorithm is also applicable to the slack-rescaling formulation of the structural SVM optimization problem.

Recently there are also works on improving the cutting plane algorithm for SVM training [49], and even non-convex bundle methods [42] have been modified for non-convex regularized risk minimization.

CHAPTER 4

STRUCTURAL SVMS FOR PROTEIN SEQUENCE ALIGNMENTS

In the last two chapters we introduced structural support vector machines and the cutting plane algorithm for solving the associated optimization problem during training. We used the part-of-speech tagging problem as a running example in Chapter 2 to illustrate how we can model the problem discriminatively with structural SVMs instead of the more traditional approach of generative modeling with hidden Markov models. In this chapter we are going to demonstrate the advantages of discriminative modeling with structural SVMs using a protein alignment application. Aligning protein sequences is an important step in the homology modeling of protein structures, but is in general difficult for protein sequence pairs with low sequence similarities. To improve the alignment accuracies of such difficult sequence pairs we need to make use of extra structural information such as secondary structures and solvent accessibility. Discriminative modeling with structural SVMs allows these extra features to be incorporated easily into the learning of alignment models. Experimental results show that by combining features judiciously, we can learn highly accurate alignment models with hundreds of thousands of parameters. These alignment models learned with structural SVMs have significantly improved alignment accuracies over state-of-art generative models.

4.1 The Protein Sequence Alignment Problem

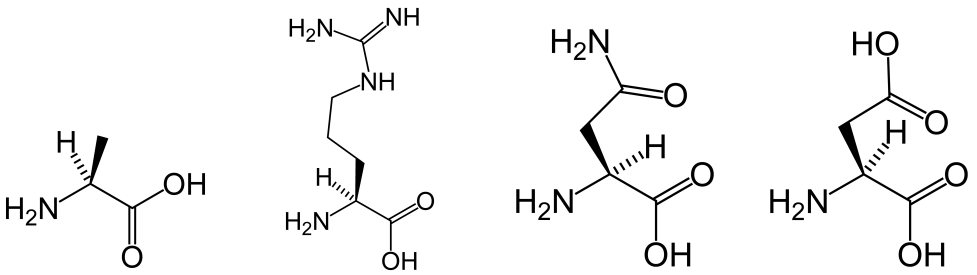
Proteins are long sequences of amino acids. There are 20 common amino acids (see Figure 4.1). Each amino acid has a carboxyl group ($COOH$ group in Figure 4.1) and an amino group (NH_2 group in Figure 4.1) that can link together to

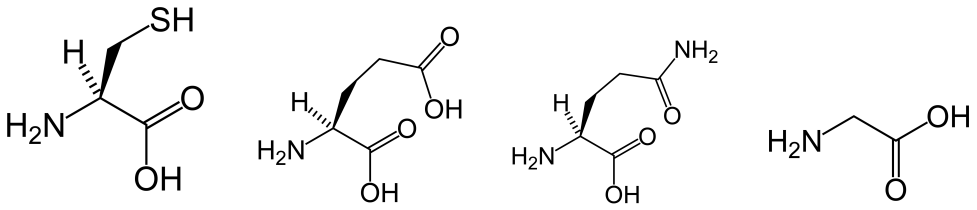
form a peptide bond. Apart from the common carboxyl and amino groups each amino acid has a side chain, which determines its unique identity.

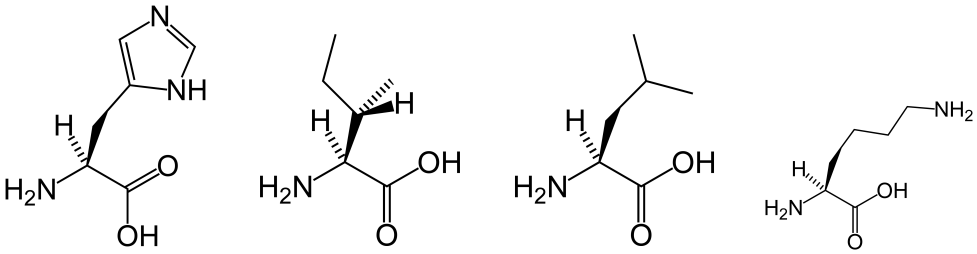
Using these peptide bonds, amino acids can form chains of length of under one hundred up to several thousands amino acids long. These amino acid chains fold (protein folding) into different 3-dimensional shapes to form different stable protein structures. With all these different shapes proteins can perform a large variety of functions in our body, such as catalyzing and inhibiting chemical reactions, trasmitting signals, or becoming structural building blocks for our muscle and hair.

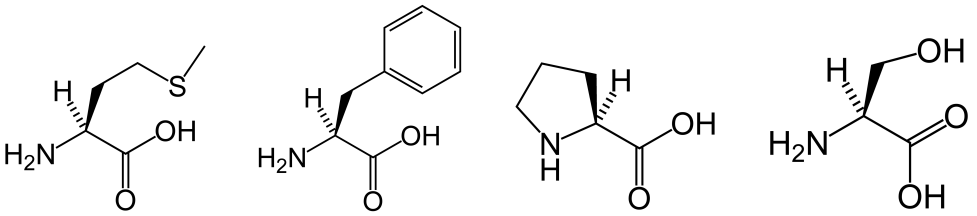
To understand how proteins work it is important to know their structures, and yet current experimental methods for protein structure determination such as Nuclear Magnetic Resonance (NMR) crystallography are very expensive and time consuming. On the other hand high-throughput genome sequencing provides us with a lot of data on protein sequences, so it will be very desirable if we could predict protein structures from the amino acid sequence alone. One possible way is to perform simulation of the molecular dynamics during protein folding, but currently even with massively parallel supercomputers we can only simulate a tiny fraction of the time required for a protein to fold (except for small proteins).

Biologists observe that in practice the number of protein folds is much less varied than their sequence compositions suggest, and protein sequences with rather different amino acid sequences can fold into similar shapes. It is hypothesized that there is only a limited number of folds in nature. This opens up another avenue of attack in predicting protein folds, which is called homology modeling. When we are given a new protein sequence with unknown struc-









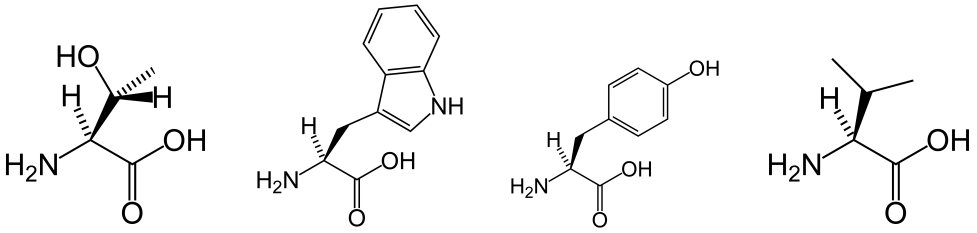


Figure 4.1: The 20 Common Amino Acids

tures, we can search in a database of proteins with known structures (such as the Protein Data Bank (PDB) [12]) to find out proteins that could potentially have similar structures with the new one. Then we align the new protein sequence to the candidate set of matching proteins in the database. With the alignments and the known 3-dimensional structures of the candidate proteins as scaffolds we can build a rough estimate of the structure of the new protein. After that further optimizations using local search can be done to optimize the side chain placements in the new structure. The alignments with known proteins in the database are very important in this procedure as they vastly reduce the size of the search space we need to work at (from a global fold search problem to a local search to improve a rough fold estimate).

It is important to obtain highly accurate alignments with candidate proteins before the model building steps since any errors introduced at this stage is difficult to correct. Yet the quality of alignments is highly dependent on the score matrices which are difficult to tune in practice. Unlike DNA sequence alignments we have a lot of structural information such as secondary structures or water accessibility in proteins, making this an ideal application scenario for discriminative modeling with structural SVMs.

4.2 Sequence Alignments and the Inverse Alignment Problem

Before discussing how we could model the problem using structural SVMs let us first have a brief review on the algorithms for calculating sequence alignments.

4.2.1 Dynamic Programming for Sequence Alignments

Computing sequence alignment is very similar to computing string edit distance, which can be formalized as follows. Given two sequences $\boldsymbol{s} = s_1 s_2 \dots s_m$ and $\boldsymbol{t} = t_1 t_2 \dots t_n$ coming from the alphabet $\mathcal{A}$, the string edit distance between $\boldsymbol{s}$ and $\boldsymbol{t}$ is the minimum number of insertion, deletion or substitution we need to transform the string $\boldsymbol{s}$ to string $\boldsymbol{t}$. Insertion is adding any $a \in \mathcal{A}$ between s_i and s_{i+1}, while deletion is the removal of any s_i in $\boldsymbol{s}$. Substitution is changing any s_i to another element a in the alphabet $\mathcal{A}$. For example, the string edit distance between $\boldsymbol{s} =' discreet'$ and $\boldsymbol{t} =' discrete'$ is 2. This can be obtained either by subsituting $s_7 =' e'$ with $'t'$ and $s_8 =' t'$ with $'e'$, or by inserting a $'t'$ between $s_6 =' e'$ and $s_7 =' e'$, and then deleting the last $'t'$ at s_8. Both of these give a string edit distance of 2. We can represent the first transformation as

$$\begin{bmatrix} d & i & s & c & r & e & e & t \\ d & i & s & c & r & e & t & e \end{bmatrix}$$

and the second transformation as

$$\begin{bmatrix} d & i & s & c & r & e & - & e & t \\ d & i & s & c & r & e & t & e & - \end{bmatrix}.$$

The "-" symbols in the first and second sequence represents insertion and deletions respectively, and we call these string edit transformations *sequence alignments*.

String edit distance can be weighted and each insertion, deletion and substitution operation can have a cost associated with it. In protein sequence alignment, a commonly used matrix is the BLOSUM62 (Figure 4.2), which comes from log-odds estimates of blocks of aligned protein sequences.

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	4	0	-2	-1	-2	0	-2	-1	-1	-1	-1	-2	-1	-1	-1	1	0	0	-3	-2
C	0	9	-3	-4	-2	-3	-3	-1	-3	-1	-1	-3	-3	-3	-3	-1	-1	-1	-2	-2
D	-2	-3	6	2	-3	-1	-1	-3	-1	-4	-3	1	-1	0	-2	0	-1	-3	-4	-3
E	-1	-4	2	5	-3	-2	0	-3	1	-3	-2	0	-1	2	0	0	-1	-2	-3	-2
F	-2	-2	-3	-3	6	-3	-1	0	-3	0	0	-3	-4	-3	-3	-2	-2	-1	1	3
G	0	-3	-1	-2	-3	6	-2	-4	-2	-4	-3	0	-2	-2	-2	0	-2	-3	-2	-3
H	-2	-3	-1	0	-1	-2	8	-3	-1	-3	-2	1	-2	0	0	-1	-2	-3	-2	2
I	-1	-1	-3	-3	0	-4	-3	4	-3	2	1	-3	-3	-3	-3	-2	-1	3	-3	-1
K	-1	-3	-1	1	-3	-2	-1	-3	5	-2	-1	0	-1	1	2	0	-1	-2	-3	-2
L	-1	-1	-4	-3	0	-4	-3	2	-2	4	2	-3	-3	-2	-2	-2	-1	1	-2	-1
M	-1	-1	-3	-2	0	-3	-2	1	-1	2	5	-2	-2	0	-1	-1	-1	1	-1	-1
N	-2	-3	1	0	-3	0	1	-3	0	-3	-2	6	-2	0	0	1	0	-3	-4	-2
P	-1	-3	-1	-1	-4	-2	-2	-3	-1	-3	-2	-2	7	-1	-2	-1	-1	-2	-4	-3
Q	-1	-3	0	2	-3	-2	0	-3	1	-2	0	0	-1	5	1	0	-1	-2	-2	-1
R	-1	-3	-2	0	-3	-2	0	-3	2	-2	-1	0	-2	1	5	-1	-1	-3	-3	-2
S	1	-1	0	0	-2	0	-1	-2	0	-2	-1	1	-1	0	-1	4	1	-2	-3	-2
T	0	-1	-1	-1	-2	-2	-2	-1	-1	-1	-1	0	-1	-1	-1	1	5	0	-2	-2
V	0	-1	-3	-2	-1	-3	-3	3	-2	1	1	-3	-2	-2	-3	-2	0	4	-3	-1
W	-3	-2	-4	-3	1	-2	-2	-3	-3	-2	-1	-4	-4	-2	-3	-3	-2	-3	11	2
Y	-2	-2	-3	-2	3	-3	2	-1	-2	-1	-1	-2	-3	-1	-2	-2	-2	-1	2	7

Figure 4.2: BLOSUM62 substitution matrix

Insertions and deletions are treated symmetrically in protein sequence alignment and they are called gaps. Given a substitution matrix $\boldsymbol{A}$ such as BLOSUM62 matrix and a gap penalty d we can find out the highest scoring alignment using dynamic programming by taking advantage of the optimal substructure in the problem. Let $F(i, j)$ be the score of the optimal alignment between subsequence $\boldsymbol{s}[1 : i]$ and $\boldsymbol{t}[1 : j]$. It can be obtained in 3 possible cases if we look at the end of the alignment. The first case is when s_i is matched with t_j, and the optimal score is the cost of substitution $\boldsymbol{A}(s_i, t_j)$ plus the score of the optimal alignment between $\boldsymbol{s}[1 : i-1]$ and $\boldsymbol{t}[1 : j-1]$, i.e., $F(i-1, j-1)$. The second case is an insertion in which t_j is inserted, and the optimal score is the score of the optimal alignment between $\boldsymbol{s}[1 : i]$ and $\boldsymbol{t}[1 : j-1]$ minus the gap penalty d. The third and final case is a deletion in which s_i is deleted, and the optimal

score is the score of the optimal alignment between $\boldsymbol{s}[1:i-1]$ and $\boldsymbol{t}[1:j]$ minus the gap penalty d. These three cases give us the following recurrence:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + \boldsymbol{A}(s_i, t_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases} \tag{4.1}$$

Thus we can fill up a two-dimensional table for $F(i,j)$ for $1 \leq i \leq m$ and $1 \leq j \leq n$, starting with the base case $F(0,0) = 0$. The score of the optimal alignment is $F(m,n)$. The optimal alignment can be recovered by storing a back pointer on where the optimal score of the current cell $F(i,j)$ is derived from in Equation (4.1). The computational complexity of this algorithm is $O(mn)$. Figure 4.3 shows how the recurrence is used to fill in the dynamic programming table around the (i,j)th entry.

This algorithm is called the Needleman-Wunsch algorithm [95] for global alignment since whole sequences are aligned. However when aligning biological sequences such as proteins or DNA it is usually more appropriate to consider the similarities between *s*ubsequences instead. Two long sequences that are dissimilar on average (long string edit distance/low alignment score) can have short subsequences that are highly similar and perform similar biological functions. It is important to detect these highly similar subsequences due to their biological significance. To detect these we have *local alignments* of sequences, which are the highest scoring alignment between subsequences of $\boldsymbol{s}$ and $\boldsymbol{t}$. Similar to global alignments we can use dynamic programming to compute the highest scoring local alignment between two sequences. Let $F(i,j)$ the highest scoring alignment between subsequences $\boldsymbol{s}[i':i]$ and $\boldsymbol{t}[j':j]$ for all $i' \leq i$

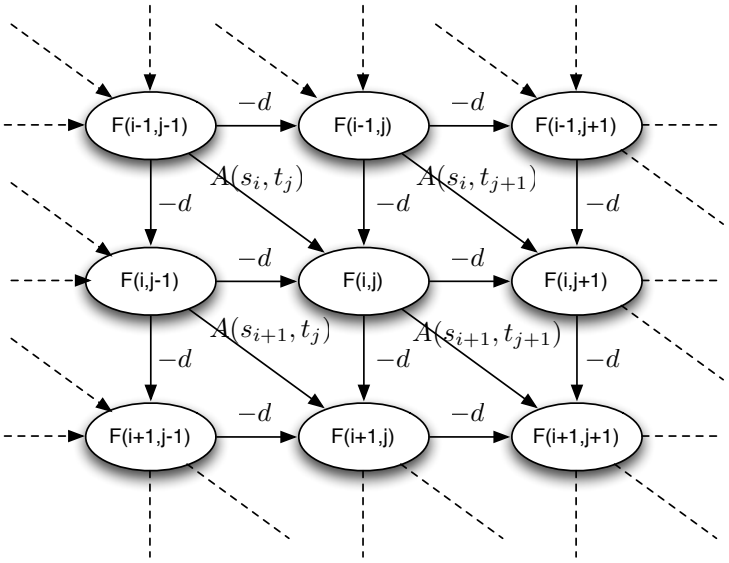


Figure 4.3: Dynamic Programming Table for Sequence Alignments

and all $j' \leq j$. We have the following recurrence equations:

$$F(i,j) = \max \begin{cases} 0 \\ F(i-1, j-1) + \boldsymbol{A}(s_i, t_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases} \tag{4.2}$$

This is called the Smith-Waterman algorithm for local alignments [118]. The recurrence equation is largely similar to Equation (4.1), with the extra option of 0 for starting the local alignment at (i, j). The score of the best local alignment can be found by searching for the position (i, j) that maximizes $F(i, j)$ in the dynamic programming table, and the local alignment can also be obtained by following appropriate back pointers. The computational complexity of this algorithm is the same as the global alignment version, and is $O(mn)$.

4.2.2 The Inverse Alignment Problem

Now we know how to align sequences using the Smith-Waterman algorithm, how can we come up with the substitution matrix $\boldsymbol{A}$ and gap penalty d so that the highest scoring alignments are the "good" alignments that we want? This parameter estimation problem is usually called the *inverse alignment problem*.

For traditional substitution cost matrices such as the BLOSUM and PAM matrices [39, 58], they are estimated by maximizing the log-odds in blocks of aligned protein sequences, manually aligned by biologists. But these are rather ad-hoc and there is no good way to include the gap penalty in the estimation process. Hidden Markov models for sequence alignments [43] (or pair-HMM) is a much more well-founded probabilistic model for the estimation of these substitution matrices and gap penalties. Like the example we had in Chapter 2 for part-of-speech tagging, it is a generative model and the joint distribution $P(x, y)$ is estimated. In this case x is the pair of amino acid sequences $(\boldsymbol{s}, \boldsymbol{t})$ and y is their alignment. Parameters can be estimated via maximum likelihood and this reduces to simple counting in the fully observed case. However in the case of protein sequence alignments we also have a lot of extra features such as secondary structures and solvent accessibilities in the input sequence pairs $x = (\boldsymbol{s}, \boldsymbol{t})$. It is difficult for generative models such as HMM to handle them without making simplifying conditional independence assumptions, such as the the secondary structure at a position being independent of the amino acid identity given the alignment operation, which is clearly not true in practice. To utilize all these information in building accurate alignment models, we shall now turn to structural SVMs to see how these extra features can be incorporated in a discriminative model in a well-founded way.

4.3 Learning to Align with Structural SVMs

Recall from Chapter 2 that structural SVMs help us to learn a parameter vector $\boldsymbol{w}$ for scoring and computing the best structured output y for a given input x:

$$y = \text{argmax}_{\hat{y} \in \mathcal{Y}} \, \boldsymbol{w} \cdot \boldsymbol{\Phi}(x, \hat{y}).$$

Let us now look at how we can encode the problem of learning the substitution matrix and the gap penalty in protein sequence alignment using the joint feature map $\boldsymbol{\Phi}$, and also how the argmax computation is related to the Smith-Waterman algorithm.

Here we make the important assumption that the alignment score is a linear sum of the scores of individual alignment operations:

$$y = \text{argmax}_{\hat{y} \in \mathcal{Y}} \, \boldsymbol{w} \cdot \sum_{k=1}^{|\hat{y}|} \boldsymbol{\phi}(x, \hat{y}^k),$$

where

$$\boldsymbol{w} \cdot \boldsymbol{\phi}(x, \hat{y}^k) = \begin{cases} \boldsymbol{A}(s_i, t_j) & \text{if } \hat{y}^k = (s_i, t_j) \text{ [a match]} \\ -d & \text{if } \hat{y}^k = (s_i, -) \text{ [an insertion]} \\ -d & \text{if } \hat{y}^k = (-, t_j) \text{ [a deletion]} \end{cases}, \tag{4.3}$$

and $\hat{y}^k$ is the kth alignment operation in $\hat{y}$. Notice that the linearity assumption allows us to continue to use the Smith-Waterman algorithm to compute the highest scoring alignment. In this case we would have a 401-dimensional feature vector $\boldsymbol{\Phi}$ (substitution matrix with 20×20 paramters with 1 gap penalty paramter). Of course this is a very simple feature vector that doesn't take full advantage of the flexibility of discriminative modeling, but we will discuss in greater details the variety of alignment model features we can use for $\boldsymbol{\phi}(x, \hat{y}^k)$ in Section 4.4.

Now we can go back to the optimization problem for structural SVM we introduced in Chapter 2.

Optimization Problem.

$$\begin{aligned}
&\min_{\boldsymbol{w},\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\
&s.t.\ \forall i, \forall \bar{y} \in \mathcal{Y}, \bar{y} \neq y_i \\
&\qquad \boldsymbol{w}\cdot\Phi(x_i, y_i) - \boldsymbol{w}\cdot\Phi(x_i, \bar{y}) \geq \Delta(y_i, \bar{y}) - \xi_i \\
&\qquad \forall i, \xi_i \geq 0
\end{aligned}$$

The meaning of the constraints in this quadratic program is clear after our discussion on feature vector encoding. It says the score of the correct alignment $\boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, y_i)$ has to be greater than the score of any other alignment $\boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i, \bar{y})$, by a margin measured by the loss function $\Delta(y_i, \bar{y})$. If the constraints cannot be satisfied then it incurs a loss through the slack variables ξ_i. The objective of the optimization problem is simply the sum of the loss $\sum_i \xi_i$ and the regularizer $1/2\|\boldsymbol{w}\|^2$ for capacity control to prevent overfitting. The parameter C controls the trade-off betweening having a lower loss on the training set and a larger margin to prevent overfitting. What remains is to find a suitable loss function Δ for our sequence alignment application.

4.3.1 Loss Functions

In this protein sequence alignment application we propose using the Q-loss function. In protein sequence alignments we are much more concerned about matched positions in the amino acid sequence (substitutions) rather than regions of insertions or deletions, which are usually loop regions that are less

Reference alignment:

$$\begin{pmatrix} - & \mathtt{A} & \mathtt{E} & \mathtt{C} & \mathtt{D} \\ \mathtt{E} & \mathtt{A} & \mathtt{C} & \mathtt{C} & - \end{pmatrix}$$

Alternative alignment:

$$\begin{pmatrix} \mathtt{A} & - & \mathtt{E} & \mathtt{C} & \mathtt{D} \\ \mathtt{E} & \mathtt{A} & \mathtt{C} & \mathtt{C} & - \end{pmatrix}$$

Figure 4.4: Illustration of Q-loss: out of three "match" operations in the reference alignment, the alternative alignment agrees on two of them. The Q-loss is therefore $1/3$.

Reference alignment:

$$\begin{pmatrix} - & \mathtt{A} & \mathtt{E} & \mathtt{C} & \mathtt{D} \\ \mathtt{E} & \mathtt{A} & \mathtt{C} & \mathtt{C} & - \end{pmatrix}$$

Alternative alignment:

$$\begin{pmatrix} \mathtt{A} & - & \mathtt{E} & \mathtt{C} & \mathtt{D} \\ \mathtt{E} & \mathtt{A} & \mathtt{C} & \mathtt{C} & - \end{pmatrix}$$

Figure 4.5: Illustration of Q4-loss: out of three "match" operations in the reference alignment, the alternative alignment has all three of them aligned to another residue within shift 4. The Q4-loss is therefore 0.

conserved structurally. To reflect this the Q-loss counts the number of incorrect matches, divided by the total number of match operations in the correct alignment y_i:

$$\Delta_Q(y_i, \hat{y}) = 1 - \sum_{k=1}^{|\hat{y}|} \delta_Q(y_i, \hat{y}^k)$$

The function $\delta_Q(y_i, \hat{y}^k)$ returns $1/M$ when $\hat{y}^k$ is a match contained in the correct alignment y_i, and 0 otherwise (including the case when $\hat{y}^k$ is a gap).

We also consider a relaxed version of the Q-loss, which we called the Q4-loss. With Q4-loss we count a match operation as correct even if it is slightly shifted, and in particular no more than 4 positions. Suppose we have an alignment operation $\hat{y}^k = (s_i, t_j)$ in $\hat{y}$. The shift of $\hat{y}^k$ is less than 4 if there is some match

operation $y^l = (s_u, t_j)$ in y with $|i - u| \leq 4$. We are interested in this relaxed loss function becaues the protein alignments in our training set is in general very noisy. Similar to the Q-loss the Q4-loss can also be written as:

$$\Delta_Q(y_i, \hat{y}) = 1 - \sum_{k=1}^{|\hat{y}|} \delta_{Q4}(y_i, \hat{y}^k),$$

where $\delta_{Q4}(y_i, \hat{y}^k)$ is $1/M$ if $\hat{y}^k$ is a match operation contained in the correct alignment y with a shift of 4 or less, and 0 otherwise.

Notice that both the Q-loss and the Q4-loss are linearly decomposable in the individual alignment operations $\hat{y}^k$. This gives us important advantages when solving the optimization problem in training.

Loss-Augmented Inference

The most important and time-consuming step in the cutting plane algorithm introduced in Chapter 3 is the computation of the argmax

$$\bar{y}_i = \text{argmax}_{\bar{y} \in \mathcal{Y}} [\Delta(y_i, \bar{y}) + \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y})].$$

Recall that we can use the Smith-Waterman algorithm to compute the highest scoring alignment if the feature vector $\boldsymbol{\Phi}$ decomposes into individual alignment operations:

$$\bar{y}_i = \text{argmax}_{\bar{y} \in \mathcal{Y}} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \bar{y}) = \text{argmax}_{\bar{y} \in \mathcal{Y}} \sum_{k=1}^{|\bar{y}|} \boldsymbol{w} \cdot \boldsymbol{\phi}(x_i, \bar{y}^k) \tag{4.4}$$

By designing the loss functions Q-loss and Q4-loss so that they are also linearly decomposable in the alignment operations, we can rewrite the argmax

computation as:

$$\begin{aligned}\bar{y}_i &= \text{argmax}_{\bar{y}\in\mathcal{Y}}[\Delta_Q(y_i,\bar{y}) + \boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i,\bar{y})] \\ &= \text{argmax}_{\bar{y}\in\mathcal{Y}}\sum_{k=1}^{|\bar{y}|}[-\delta_Q(y_i,\bar{y}^k) + \boldsymbol{w}\cdot\boldsymbol{\phi}(x_i,\bar{y}^k)].\end{aligned}$$

It is easy to see that the argmax computation have the same form as in Equation (4.4) (with the cost of alignment operation modified), and therefore we can use the Smith-Waterman algorithm to solve it.

4.4 Aligning with Many Features

Let us now look at the main strength of discriminative modeling with structural SVMs, which is the ease and flexibility to construct new features from the input x to improve prediction accuracy. For the protein sequence alignment problem we have a set of different features for the two sequences in $x = (\boldsymbol{s}, \boldsymbol{t})$. We use $R^i_{\boldsymbol{s}}$, $S^i_{\boldsymbol{s}}$, $A^i_{\boldsymbol{s}}$ to denote the residue, secondary structure and exposed surface area at the ith position of the target sequence $\boldsymbol{s}$, and $R^j_{\boldsymbol{t}}$, $S^j_{\boldsymbol{t}}$, $A^j_{\boldsymbol{t}}$ to denote the corresponding features in the template sequence $\boldsymbol{t}$ at the jth position. Note that because the target sequence $\boldsymbol{s}$ is the new protein sequence with unknown structure, we have no actual information about the secondary structures and exposed surface area from experimental data. The values of secondary structures and exposed surface areas for the target sequence are predicted from another computer program called SABLE [1] from the residue sequence alone. The program has over 70% accuracy and so we include its output as features. The secondary structures and exposed surface areas for the template sequence are actual as we have their full structural information in the database. We also use $\mathbf{R}$, $\mathbf{S}$, $\mathbf{A}$ to denote the set of possible values for residue, secondary structure, and exposed surface area.

In the last alignment model considered we include PSI-BLAST [3] profile as an extra feature, and we denote them as $P^i_{\boldsymbol{s}}$ and $P^j_{\boldsymbol{t}}$ for the profiles at the ith and the jth positions in the target and the template sequences respectively.

4.4.1 Substitution Cost Model

For the substitution costs, we consider the following six models for $\boldsymbol{\phi}(x, \hat{y}^k)$. Since the length of alignments of different examples varies greatly, we normalize each $\boldsymbol{\phi}$ by dividing with $|\boldsymbol{s}| + |\boldsymbol{t}|$.

Simple

In this alignment model we only consider the substitution of single features. Let $\hat{y}^k = (s_i, t_j)$ be a match operation. We define $\boldsymbol{\phi}(x, \hat{y}^k)$ to be

$$
\begin{aligned}
&\boldsymbol{\phi}_{\textit{Simple}}(y^k, \boldsymbol{s}, \boldsymbol{t}) \\
&= \sum_{r_1, r_2 \in \mathbf{R}} \mathbb{I}[R^i_{\boldsymbol{s}} = r_1, R^j_{\boldsymbol{t}} = r_2] + \sum_{r_1 \in \mathbf{R}, s_2 \in \mathbf{S}} \mathbb{I}[R^i_{\boldsymbol{s}} = r_1, S^j_{\boldsymbol{t}} = s_2] + \sum_{r_1 \in \mathbf{R}, a_2 \in \mathbf{A}} \mathbb{I}[R^i_{\boldsymbol{s}} = r_1, A^j_{\boldsymbol{t}} = a_2] \\
&+ \sum_{s_1 \in \mathbf{S}, r_2 \in \mathbf{R}} \mathbb{I}[S^i_{\boldsymbol{s}} = s_1, R^j_{\boldsymbol{t}} = r_2] + \sum_{s_1, s_2 \in \mathbf{S}} \mathbb{I}[S^i_{\boldsymbol{s}} = s_1, S^j_{\boldsymbol{t}} = s_2] + \sum_{s_1 \in \mathbf{S}, a_2 \in \mathbf{A}} \mathbb{I}[S^i_{\boldsymbol{s}} = s_1, A^j_{\boldsymbol{t}} = a_2] \\
&+ \sum_{a_1 \in \mathbf{A}, r_2 \in \mathbf{R}} \mathbb{I}[A^i_{\boldsymbol{s}} = a_1, R^j_{\boldsymbol{t}} = r_2] + \sum_{a_1 \in \mathbf{A}, s_2 \in \mathbf{S}} \mathbb{I}[A^i_{\boldsymbol{s}} = a_1, S^j_{\boldsymbol{t}} = s_2] + \sum_{a_1, a_2 \in \mathbf{A}} \mathbb{I}[A^i_{\boldsymbol{s}} = a_1, A^j_{\boldsymbol{t}} = a_2] \ ,
\end{aligned}
$$

where $\mathbb{I}[\rho]$ is a function that returns a vector with '1' in the position designated to ρ if the boolean expression ρ is true, and returns '0' otherwise and in all other positions. For example, $\mathbb{I}[R^3_{\boldsymbol{s}} = \text{'A'}, S^7_{\boldsymbol{t}} = \text{'}\alpha\text{'}]$ returns '1' in the particular dimension corresponding to $\mathbb{I}[R^i_{\boldsymbol{s}} = \text{'A'}, S^j_{\boldsymbol{t}} = \text{'}\alpha\text{'}]$, if $y^k = (s_3, t_7)$ aligns the residue alanine 'A' in $\boldsymbol{s}$ with an alpha helix 'α' in $\boldsymbol{t}$. Otherwise, it returns '0' in this dimension. For all other dimensions it always returns '0'. Note that each such

dimension corresponds to a particular position in cost vector $\boldsymbol{w}$. Note also that each feature vector $\boldsymbol{\phi}_{Simple}(y^k, \boldsymbol{s}, \boldsymbol{t})$ has exactly 9 '1's corresponding to the 9 terms in the sum, and is zero elsewhere.

Anova2

In this more complex feature vector we take the interactions between pairs of structural annotations at the same position in the sequence into account. We define $\boldsymbol{\phi}(y^k, \boldsymbol{s}, \boldsymbol{t})$ to be

$$\begin{aligned}
\boldsymbol{\phi}_{Anova2}(y^k, \boldsymbol{s}, \boldsymbol{t}) = & \sum_{r_1, r_2 \in \mathbf{R}, s_1, s_2 \in \mathbf{S}} \mathbb{I}[R^i_{\boldsymbol{s}} = r_1, S^i_{\boldsymbol{s}} = s_1, R^j_{\boldsymbol{t}} = r_2, S^j_{\boldsymbol{t}} = s_2] \\
& + \sum_{\substack{r_1 \in \mathbf{R}, a_2 \in \mathbf{A}, \\ s_1, s_2 \in \mathbf{S}}} \mathbb{I}[R^i_{\boldsymbol{s}} = r_1, S^i_{\boldsymbol{s}} = s_1, S^j_{\boldsymbol{t}} = s_2, A^j_{\boldsymbol{t}} = a_2] + \sum_{\substack{r_1, r_2 \in \mathbf{R}, s_1 \in \mathbf{S}, \\ a_2 \in \mathbf{A}}} \mathbb{I}[R^i_{\boldsymbol{s}} = r_1, S^i_{\boldsymbol{s}} = s_1, A^j_{\boldsymbol{t}} = a_2, R^j_{\boldsymbol{t}} = r_2] \\
& + \sum_{\substack{r_2 \in \mathbf{R}, a_1 \in \mathbf{A}, \\ s_1, s_2 \in \mathbf{S}}} \mathbb{I}[S^i_{\boldsymbol{s}} = s_1, A^i_{\boldsymbol{s}} = a_1, R^j_{\boldsymbol{t}} = r_2, S^j_{\boldsymbol{t}} = s_2] + \sum_{\substack{s_1, s_2 \in \mathbf{S}, \\ a_1, a_2 \in \mathbf{A}}} \mathbb{I}[S^i_{\boldsymbol{s}} = s_1, A^i_{\boldsymbol{s}} = a_1, S^j_{\boldsymbol{t}} = s_2, A^j_{\boldsymbol{t}} = a_2] \\
& + \sum_{\substack{r_2 \in \mathbf{R}, s_1 \in \mathbf{S}, \\ a_1, a_2 \in \mathbf{A}}} \mathbb{I}[S^i_{\boldsymbol{s}} = s_1, A^i_{\boldsymbol{s}} = a_1, A^j_{\boldsymbol{t}} = a_2, R^j_{\boldsymbol{t}} = r_2] + \sum_{\substack{r_1, r_2 \in \mathbf{R}, s_2 \in \mathbf{S}, \\ a_2 \in \mathbf{A}}} \mathbb{I}[A^i_{\boldsymbol{s}} = a_1, R^i_{\boldsymbol{s}} = r_1, R^j_{\boldsymbol{t}} = r_2, S^j_{\boldsymbol{t}} = s_2] \\
& + \sum_{\substack{r_1 \in \mathbf{R}, s_2 \in \mathbf{S}, \\ a_1, a_2 \in \mathbf{A}}} \mathbb{I}[A^i_{\boldsymbol{s}} = a_1, R^i_{\boldsymbol{s}} = r_1, S^j_{\boldsymbol{t}} = s_2, A^j_{\boldsymbol{t}} = a_2] + \sum_{\substack{r_1, r_2 \in \mathbf{R}, \\ a_1, a_2 \in \mathbf{A}}} \mathbb{I}[A^i_{\boldsymbol{s}} = a_1, R^i_{\boldsymbol{s}} = r_1, A^j_{\boldsymbol{t}} = a_2, R^j_{\boldsymbol{t}} = r_2] \quad .
\end{aligned}$$

For example, the term $\mathbb{I}[R^i_{\boldsymbol{s}} = r_1, S^i_{\boldsymbol{s}} = s_1, R^j_{\boldsymbol{t}} = r_2, S^j_{\boldsymbol{t}} = s_2]$ returns '1' in the appropriate position, if $y^k = (s_i, t_j)$ aligns residue of type r_1 in secondary structure s_1 in the target with residue of type r_2 in secondary structure s_2 in the template. These features capture pairwise interaction of structural annotations within the same sequence.

Tensor

In this even more complex alignment model we consider the interaction of all three structural annotations. Note that there is only one non-zero feature in this feature vector.

$$\boldsymbol{\phi}_{\textit{Tensor}}(y^k, \boldsymbol{s}, \boldsymbol{t}) = \sum_{r_1,r_2 \in \mathbf{R}, s_1,s_2 \in \mathbf{S}, a_1,a_2 \in \mathbf{A}} \mathbb{I}[R^i_{\boldsymbol{s}} = r_1, S^i_{\boldsymbol{s}} = s_1, A^i_{\boldsymbol{s}} = a_1, R^j_{\boldsymbol{t}} = r_2, S^j_{\boldsymbol{t}} = s_2, A^j_{\boldsymbol{t}} = a_2]$$

Simple+Anova2+Tensor

This alignment model is the union of all features in the first three alignment models, i.e. $\boldsymbol{\phi}_{\textit{Simple}}(y^k, \boldsymbol{s}, \boldsymbol{t}) + \boldsymbol{\phi}_{\textit{Anova2}}(y^k, \boldsymbol{s}, \boldsymbol{t}) + \boldsymbol{\phi}_{\textit{Tensor}}(y^k, \boldsymbol{s}, \boldsymbol{t})$.

Window

On top of the *Simple+Anova2+Tensor* feature vector, we add several terms involving the substitution score of a sliding window of features centered around positions i and j.

$$\begin{aligned}
\boldsymbol{\phi}_{\textit{Window}}(y^k, \boldsymbol{s}, \boldsymbol{t}) = \boldsymbol{\phi}_{\textit{Simple}}(y^k, \boldsymbol{s}, \boldsymbol{t}) + \boldsymbol{\phi}_{\textit{Anova2}}(y^k, \boldsymbol{s}, \boldsymbol{t}) + \boldsymbol{\phi}_{\textit{Tensor}}(y^k, \boldsymbol{s}, \boldsymbol{t}) \\
+ \sum_{r_1,r_2,r_3 \in \mathbf{R}, r_4,r_5,r_6 \in \mathbf{R}} \mathbb{I}[R^{i-1}_{\boldsymbol{s}} = r_1, R^i_{\boldsymbol{s}} = r_2, R^{i+1}_{\boldsymbol{s}} = r_3, R^{j-1}_{\boldsymbol{t}} = r_4, R^j_{\boldsymbol{t}} = r_5, R^{j+1}_{\boldsymbol{t}} = r_6] \\
+ \sum_{s_1,\ldots,s_5 \in \mathbf{S}, s_6,\ldots,s_{10} \in \mathbf{S}} \mathbb{I}[S^{i-2}_{\boldsymbol{s}} = s_1, \ldots, S^{i+2}_{\boldsymbol{s}} = s_5, S^{j-2}_{\boldsymbol{t}} = s_6, \ldots, S^{j+2}_{\boldsymbol{t}} = s_{10}] \\
+ \sum_{a_1,\ldots,a_7 \in \mathbf{A}, a_8,\ldots,a_{14} \in \mathbf{A}} \mathbb{I}[A^{i-3}_{\boldsymbol{s}} = a_1, \ldots, A^{i+3}_{\boldsymbol{s}} = a_7, A^{j-3}_{\boldsymbol{t}} = a_8, \ldots, A^{j+3}_{\boldsymbol{t}} = a_{14}]
\end{aligned}$$

The first sliding window term counts the occurence of substituting a triplet of residues (r_1, r_2, r_3) in the target with another triplet (r_4, r_5, r_6) in the template. The other two terms counts the occurence of substitution of two windows of

secondary structures of length 5, and the occurence of substitution of two windows of surface area type of length 7 respectively. To reduce dimensionality of these features, we bin the residues into 7 groups ({A,G,P,S,T}, {C}, {D,E,N,Q}, {F,W,Y}, {H,K,R}, {I,L,M,V}, {X}, where X stands for missing value and ends of sequences), and the surface area into 2 values, exposed or buried.

Profile

Finally we explore adding profile information to the alignment model, and we use $P^i_{\boldsymbol{s}}$ to denote the profile (a real-valued vector of dimension 20 for 20 amino acids) at the ith position of the target sequence. $P^i_{\boldsymbol{s}}(r_1)$ denotes the weight assigned to amino acid r_1 by the profile $P^i_{\boldsymbol{s}}$. The definitions for profiles in the template $P^j_{\boldsymbol{t}}$ are similar. The *Profile* alignment model is very similar to the *Window* alignment model, with the single interaction component $\phi_{\textit{Simple}}$ replaced by a version that includes profile information $\phi_{\textit{SimpleWithProfile}}$:

$$
\begin{aligned}
&\phi_{\textit{SimpleWithProfile}}(y^k, \boldsymbol{s}, \boldsymbol{t}) \\
=\ &\phi_{\textit{Simple}}(y^k, \boldsymbol{s}, \boldsymbol{t}) \\
&+\sum_{r_1 \in \mathbf{R}, r_2 \in \mathbf{R}} \mathbb{I}[R^i_{\boldsymbol{s}} = r_1] P^j_{\boldsymbol{t}}(r_2) + \sum_{s_1 \in \mathbf{S}, r_2 \in \mathbf{R}} \mathbb{I}[S^i_{\boldsymbol{s}} = s_1] P^j_{\boldsymbol{t}}(r_2) + \sum_{a_1 \in \mathbf{A}, r_2 \in \mathbf{R}} \mathbb{I}[A^i_{\boldsymbol{s}} = a_1] P^j_{\boldsymbol{t}}(r_2) \\
&+\sum_{r_1 \in \mathbf{R}, r_2 \in \mathbf{R}} P^i_{\boldsymbol{s}}(r_1) \mathbb{I}[R^j_{\boldsymbol{t}} = r_2] + \sum_{r_1 \in \mathbf{R}, s_2 \in \mathbf{S}} P^i_{\boldsymbol{s}}(r_1) \mathbb{I}[S^j_{\boldsymbol{t}} = s_2] + \sum_{r_1 \in \mathbf{R}, a_2 \in \mathbf{A}} P^i_{\boldsymbol{s}}(r_1) \mathbb{I}[A^j_{\boldsymbol{t}} = a_2] \\
&+\sum_{r_1 \in \mathbf{R}, r_2 \in \mathbf{R}} P^i_{\boldsymbol{s}}(r_1) P^j_{\boldsymbol{t}}(r_2)
\end{aligned}
$$

We only incorporate profile information into the single interaction component $\phi_{\textit{Simple}}$ instead of higher-order interaction components like $\phi_{\textit{Anova2}}$ since our goal is to test the usefulness of including profile information in our alignment models, but not to find the best alignment model that captures all the interactions

between all features. This alignment model can be expressed as:

$$\begin{aligned}\boldsymbol{\phi}_{\textit{Profile}}(y^k, \boldsymbol{s}, \boldsymbol{t}) = \boldsymbol{\phi}_{\textit{SimpleWithProfile}}(y^k, \boldsymbol{s}, \boldsymbol{t}) + \boldsymbol{\phi}_{\textit{Anova2}}(y^k, \boldsymbol{s}, \boldsymbol{t}) + \boldsymbol{\phi}_{\textit{Tensor}}(y^k, \boldsymbol{s}, \boldsymbol{t}) \\ + \sum_{r_1, r_2, r_3 \in \mathbf{R}, r_4, r_5, r_6 \in \mathbf{R}} \mathbb{I}[R_{\boldsymbol{s}}^{i-1} = r_1, R_{\boldsymbol{s}}^{i} = r_2, R_{\boldsymbol{s}}^{i+1} = r_3, R_{\boldsymbol{t}}^{j-1} = r_4, R_{\boldsymbol{t}}^{j} = r_5, R_{\boldsymbol{t}}^{j+1} = r_6] \\ + \sum_{s_1, ..., s_5 \in \mathbf{S}, s_6, ..., s_{10} \in \mathbf{S}} \mathbb{I}[S_{\boldsymbol{s}}^{i-2} = s_1, ..., S_{\boldsymbol{s}}^{i+2} = s_5, S_{\boldsymbol{t}}^{j-2} = s_6, ..., S_{\boldsymbol{t}}^{j+2} = s_{10}] \\ + \sum_{a_1, ..., a_7 \in \mathbf{A}, a_8, ..., a_{14} \in \mathbf{A}} \mathbb{I}[A_{\boldsymbol{s}}^{i-3} = a_1, ..., A_{\boldsymbol{s}}^{i+3} = a_7, A_{\boldsymbol{t}}^{j-3} = a_8, ..., A_{\boldsymbol{t}}^{j+3} = a_{14}]\end{aligned}$$

4.4.2 Gap Cost Model

All alignment models above share the following gap model. Consider the cost of opening a gap between position i and $i+1$ in the target sequence $\boldsymbol{s}$ against position j in the template structure $\boldsymbol{t}$, as depicted by the following diagram

Target	s_i	–	–	$\cdots$	–	s_{i+1}
Template	$\cdots$	t_j	t_{j+1}	$\cdots$	t_{j+k}	$\cdots$

We allow the cost of opening a gap to depend on the structural type at position j in the template structure. It also depends on the structural type of the target sequence immediately before the gap at position i as well as the structural type immediately after the gap at position $i+1$. Suppose y^k is a gap operation that opens a gap between position i and $i+1$ in the target against position j in the template sequence. The feature vector for this gap operation is:

$$\begin{aligned}\phi_{\textit{Gap}}(y^k, \boldsymbol{s}, \boldsymbol{t}) = \sum_{r_1 \in R} \mathbb{G}[R_{\boldsymbol{t}}^{j} = r_1] + \sum_{s_1 \in S, a_1 \in A} \mathbb{G}[S_{\boldsymbol{t}}^{j} = s_1, A_{\boldsymbol{t}}^{j} = a_1] \\ + \sum_{s1, s2 \in S, a_1, a_2 \in A} \mathbb{G}[S_{\boldsymbol{s}}^{i} = s_1, A_{\boldsymbol{s}}^{i} = a_1, S_{\boldsymbol{s}}^{i+1} = s_2, A_{\boldsymbol{s}}^{i+1} = a_2]\end{aligned}$$

$\mathbb{G}$ is analogous to $\mathbb{I}$, but we use a different symbol to indicate that it maps to a different set of dimensions. The first two terms create features for the residue

types and joint features of secondary structure with exposed surface area at t_j. The term $\mathbb{G}[S^i_{\boldsymbol{s}} = s_1, A^i_{\boldsymbol{s}} = a_1, S^{i+1}_{\boldsymbol{s}} = s_2, A^{i+1}_{\boldsymbol{s}} = a_2]$ considers the structure before and after the gap. For example, $\mathbb{G}[S^i_{\boldsymbol{s}} = \text{`}\alpha\text{'}, A^i_{\boldsymbol{s}} = \text{`0'}, S^{i+1}_{\boldsymbol{s}} = \text{`}\alpha\text{'}, A^{i+1}_{\boldsymbol{s}} = \text{`1'}]$ maps to the dimension for the cost of opening a gap between a position in an alpha-helix of surface type 0 with a consecutive position in the alpha-helix with surface type 1.

The case of opening a gap in the template involves exactly the same costs, with the role of target and template reversed.

4.5 Experiments

4.5.1 Data

Training and Validation Sets

The training set and the validation data for parameter tuning are the same as those used in [104]. The data set contains 1379 target sequences, and each target sequence $\boldsymbol{s}$ has one or more template structures $\boldsymbol{t}$ associated with it. Structural alignments between target and template are generated using the Combinatorial Extension (CE) program [117], which approximately minimize the root-mean-squared-distance (rmsd) between the target and template structures by joining and extending similar fragment pairs. All the alignments between the target and template have CE Z-score of at least 4.5, an indication that the target and template have high structural similarities. The dataset is randomly split into two sets, with a training set consisting of 690 targets and a validation set contain-

ing 689 targets. The resulting training set contains 4542 examples (i.e., pairwise alignments) while the validation set contains 4587 examples. These structural alignments are used as our ground truth labels y.

To realistically evaluate our alignment models when employed in homology modeling we removed all structural information in the target sequence $\boldsymbol{s}$ in the input $x = (\boldsymbol{s}, \boldsymbol{t})$. The template sequence have the residue sequence information, and actual structural annotations on secondary structures and relative exposed surface areas, computed by the program DSSP [71]. For the templates, the secondary structures are binned into 5 types while the exposed surface areas are binned into 6 types. The target sequences only have the residue sequence information, but we generate extra features by using the SABLE program [1] to predict the secondary structure and relative exposed surface area from the residue information. Although these structural annotations are not actual they still have about 70% accuracy [1]. For the targets, the secondary structures are binned into 3 types and the relative exposed surface areas are binned into 4 types. In the last alignment model we also include evolutionary information in the form of PSI-BLAST profile to investigate its usefulness in alignment. PSI-BLAST profile is a position-specific score matrix constructed from the substitution counts of similar sequences collected using a PSI-BLAST search in a large database.

Test Sets

The test set is based on a database of protein structures that is used by the modeling program LOOPP (`http://cbsuapps.tc.cornell.edu/loopp.aspx`). We selected 4185 structures from the new PDB structures released between June 2005 and June 2006 via clustering. These structures serve as target sequences in

our test set and none of them appear in the training or validation sets since the training and validation sets were developed before June 2005. Each of these 4185 structures is aligned against all other structures using the structural alignment program TM-align [157]. Pairs that score 0.5 or better are considered homologous and are added to the test set. The selected pairs are then aligned by the structural alignment program CE. Only alignments that have CE Z-score higher than 4.5 are included in the final test set, providing a total of 29345 alignments to consider. The features generated for the test set are exactly the same as those in the training set.

4.5.2 Results

Table 4.1 shows the Q-scores of the different alignment models trained with the structural SVM algorithm using Q-loss. As decribed above, we report the five-fold cross validation results for the value of C that optimizes performance on the validation set. The table also shows the number of features in each model. Note that the training and the validation set are composed of more difficult cases than the test set, which explains the generally higher Q-scores on the test set.

Table 4.1 shows a general trend that as we add features carefully the performance of the alignment models increases. The *Simple* alignment model is too simple to fit the training data, indicated by the low Q-score on the training set. The weight vector obtained in this low dimension feature space is also not stable, as the standard error from the five-fold cross validation estimates are much higher than the other alignment models. The more expressive *Anova2* model leads to substantial improvement in Q-score over *Simple*. This shows that con-

Table 4.1: Q-score of the SVM algorithm for different alignment models (average of 5CV, standard error in brackets).

	# Features	Training	Validation	Test
Simple	1020	31.93	28.53 (2.34)	42.08 (2.02)
Anova2	49634	41.25	33.73 (0.87)	44.57 (0.78)
Tensor	203280	55.27	33.20 (0.78)	42.41 (0.18)
Simple+Anova2+Tensor	253934	47.77	34.39 (0.55)	44.59 (0.28)
Window	447016	44.40	35.23 (1.05)	46.34 (0.70)
Profile	448642	54.58	38.50 (0.51)	49.94 (0.11)

sidering pairwise interaction between structural annotations is meaningful. The *Tensor* alignment model does worse than *Anova2* on the test set but has similar Q-score on the validation set. There are signs of overfitting in the relatively high Q-score on the training set. Adding the substitution costs together, as in *Simple+Anova2+Tensor*, does not give us much improvement in accuracy. The performance of the model is very close to *Anova2*. Only when we incorporate structural information in the local neighbourhood, as in the alignment model *Window*, do we see another jump in the Q-score on the test set. Adding evolutionary information in the form of PSI-BLAST profile further enhances the performance as in the alignment model *Profile*, which has Q-scores close to 50 on the test set. The Q-score of 49.94 in the alignment model *Profile* is substantially better than the Q-score of 42.08 of the *Simple* alignment model that we started with. To provide a baseline, the Q-score of BLAST is 25.88 on the test set.

Table 4.2: Comparing training for Q-score with training for Q_4-score by test set performance.

Anova2	test Q	test Q4
train Q	44.57 (0.77)	66.30 (0.33)
train Q4	46.32 (0.06)	68.87 (0.44)

Window	test Q	test Q4
train Q	46.34 (0.70)	67.58 (1.30)
train Q4	47.31 (0.21)	70.52 (0.33)

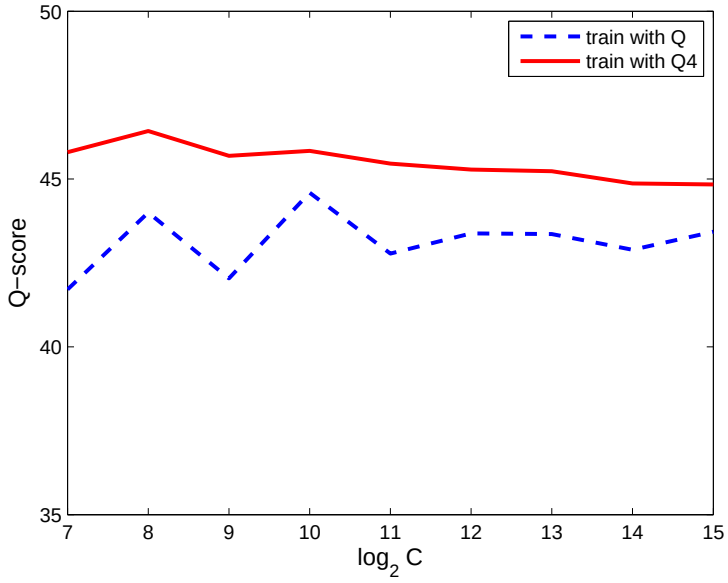


Figure 4.6: Q-score of the *Anova2* alignment model against C.

Q-Loss VS Q4-loss Functions

The structural SVM method allows the use of different loss functions during training. The Q-loss used in the previous subsection is rather stringent and does not necessarily summarize the quality of an alignment well. For example, if all the aligned positions are shifted by just 1, the Q-loss will jump from 0 to 1, which is roughly the same Q-loss as that of a completely random align-

ment. Furthermore, the Q-loss does not account for the approximate nature of the training alignments, since there is typically no single exact alignment in sequence to structure alignment that is clearly correct.

Instead of Q-loss, we now consider the Q4-loss function. Q4-loss counts a residue as correctly aligned if the shift from its position in the reference alignment is no more than 4. The Q4-loss function captures our intuition that small shifts in alignment could be tolerated, and such alignments should be differentiated from alignments that are completely wrong. We repeat our experiments on two alignment models from the last section, *Anova2* and *Window*, but this time we train them with Q4 as the loss function. The results on the test set are shown in Table 4.2. For each table entry, we select C on the validation set with respect to the performance measure that is reported. As before, the Q-scores and standard errors are estimated using models trained from five-fold cross validation.

Table 4.2 shows that the models trained on Q4-loss show better Q4-loss performance on the test set. More surprisingly, the models trained on Q4 also show better Q-score on the test set. Figure 4.6 shows the Q-score of the *Anova2* alignment model for different values of C. We can see that models trained on the Q4-loss function dominate those trained on the Q-loss across the whole range of values of C. Similar trends are observed in other alignment models. This gives evidence that Q4 can indeed effectively account for the inaccuracy of the training alignments, instead of trying to model the noise. However, in situations where the alignments have higher sequence similarity or we are more confident of the alignments, the use of Q-loss or reducing the allowable shift of 4 in Q4 to lower values could be beneficial. The flexibility of structural SVM regarding the selection of loss function would cater either of these situations.

Table 4.3: Comparing training for Q-score with training for Q4-score by test set performance.

	Q on test	Q4 on test
SVM (*Window*, Q4)	47.31 (0.21)	70.53 (0.33)
SVM (*Profile*, Q4)	50.15 (0.19)	72.19 (0.26)
SSALN	47.49	67.51
BLAST	25.88	30.66
TM-align	70.21	85.45

Comparison against other methods

As selected by validation performance, the best alignment model is *Profile* trained on Q4, while the best alignment model without the use of evolutionary information is *Window* trained on Q4. Table 4.3 shows the test set performance of various other methods in comparison. SSALN [104] is one of the best current alignment algorithm trained using generative methods, and it outperforms alignment and threading algorithms like CLUSTALW [129], GenTHREADER [69], and FUGUE [116] on a variety of benchmarks. It incorporates structural information in its substitution matrices, and contains a hand-tuned gap model. The training set we use is derived from the same training set used by SSALN (we are using about 4500 alignments out of more than 5200 alignments in the SSALN training set), and the same set of structural annotations (with the exception of the last alignment model *Profile*), so a direct comparison is particularly meaningful. Using exactly the same set of features the structural SVM model *Window* substantially outperforms SSALN with respect to Q4-score, and essen-

tially ties with SSALN on Q-score. Incorporating profile information makes the SVM model *Profile* performs even better. The performance of BLAST is included to provide a baseline. The performance of the structural alignment program TM-align [157] is reported here to show its agreement with the CE alignments, and demonstrates the rather high inherent noise in the data.

4.6 Conclusions

In this chapter we explore an application of structural SVMs for learning complex alignment models for sequence to structure alignment. We show that the structural SVM algorithm can learn high dimensional alignment models that include many features beyond residue identity while effectively controlling overfitting. Unlike generative methods, it does not require independence assumptions between features. The structural SVM method provides great modeling flexibility to biologists, allowing the estimation of models that include all available information without having to worrying about statistical dependencies between features. Furthermore, we show that one can incorporate different loss functions during training, which provides the flexibility to specify the costs of different alignment errors. The empirical results show that the structural SVM algorithm outperforms one of the best current generative models, and is practical to train on large datasets.

4.7 Bibilographical Notes

This work on applying structural SVMs to learn parameters for protein alignment models was first published at [153, 152].

Closely related to our work, also from the field of machine learning, is the application of conditional random fields to learn protein alignments [41] and string edit distance [90]. Compared against CRF our structural SVM approach does not require the computation of the partition function in training, and allows extra flexibility through the use of loss functions to control the margin.

On the other hand there are also a lot of work on protein threading (sequence to structure alignment) based on mathematical programming. Our work was influenced by [92, 139], which uses linear programming to design protein folding potentials. There are also work on solving the inverse alignment problem using linear programming [73] and linear programming relaxation of mixed integer programming for protein threading [147].

The problem of inverse alignment was first formulated in [55]. They discussed inverse alignment in the context of parametric sequence alignment and identified geometric properties of the space of cost model. The work in [100] analyzed the space of models and showed that some aspects of its complexity grow only polynomially. The first concrete algorithm for inverse sequence alignment was proposed in [123]. While they proved that their algorithm finds a consistent cost model in polynomial time, their algorithm was limited to particular cost models with 3 parameters. These works focus on the theoretical aspect of the problem, while machine learning approaches like ours focus more on the practical aspect and improving the accuracies of alignment models learned.

CHAPTER 5

STRUCTURAL SVMS WITH LATENT VARIABLES

In the last chapter we demonstrated the strengths of disciminative modeling with structural SVMs by applying it to learn highly accurate alignment models with hundreds of thousands of features. We have also seen that by plugging in different joint feature maps, loss functions, and inference algorithms, it is very easy to adapt structural SVM to learn prediction models for seemingly different tasks such as part-of-speech tagging and protein sequence alignments. Despite this ease of adapting to different applications and its high prediction accuracies through incorporating many features, there is still a large class of structured output prediction problems that cannot be handled by standard structural SVM. This class of problems involves latent or missing information and is very common in natural language processing and computer vision. In this chapter we propose an extension to structural SVM that allows the incorporation of latent variables and an efficient algorithm to solve the associated training problem. Like standard structural SVMs it is very easy to adapt it to different applications, and we illustrate its power through three different applications in computational biology, natural language processing, and information retrieval.

5.1 Latent Information in Structured Output Learning

For the protein sequence alignment problem we looked at in Chapter 4, all the necessary modeling information such as the input features on amino acid, secondary structures, and output alignments are all included in the training set. We are very lucky to have all these annotation and alignment information available through the use of both automatic tools and manual curating by biologists.

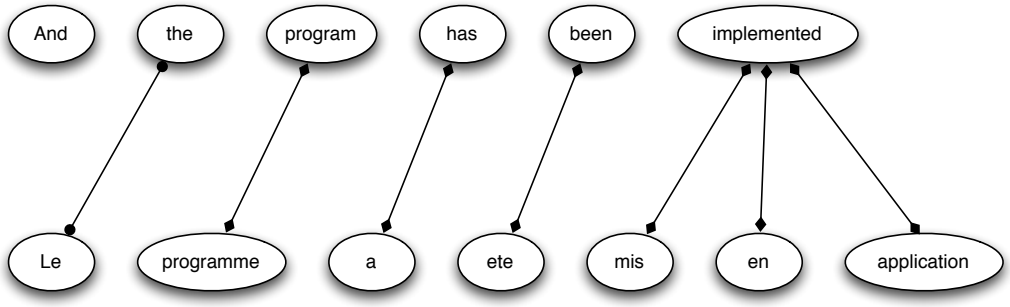


Figure 5.1: French-English word alignment example from Brown et al. [17]

However, in many structured output prediction problems important modeling information is either impossible to observe or not available during the natural data collection process.

One prime example of this is the role of word alignments in statistical machine translation. In their seminal work Brown et al. [17] proposed to learn translation of French words into English words through the use of translated French-English sentence pairs in a parallel corpus, such as the translated proceedings of the Canadian parliament. Consider the translation example of ”Le programme a ete mis en application” into ”And the program has been implemented” in [17]. We know the French word ”programme” translates into the English word ”program” and the French words ”mis en application” translate into the English word ”implemented” (see Figure 5.1). Brown et al. called these many-to-one mappings from French to English *word alignments*, and these are essential in the estimation of word translation probabilities in their generative model for machine translation. Yet none of these word alignments are observed in the training data. All we get from the parliamentary proceedings training data are sentence-to-sentence translations from French to English. The common source of training data for machine translation such as parliamentary pro-

ceedings, instruction manuals or news reports have no explicit word alignment information available. Although there have been some efforts to train better word alignment models via human annotation of the parallel corpora with word alignment information [127, 13], the availability of such data is limited due to high cost of annotation. To model the machine translation problem properly we need to model this latent word alignment information well.

Computer vision is another area that is rich with structured output prediction problems involving latent information. Take the example of recognition of cars and pedestrians in different road scenes. The training data usually consists of images of different road scenes, and the label comes in the form of bounding boxes on cars and pedestrians. In object recognition for a complex object such as human or cars it is usually helpful to know the parts of the object, such as limbs in human or the wheels in a car. However, such information is usually not present in the training data and to acquire it will require costly extra annotation effort. The best option is to model these parts of objects as latent information in our learning problem.

5.2 Structural SVMs with Latent Variables

How can we extend our structural SVM framework to handle such latent information? In particular we would like to have an extension that is general enough to include a diverse set of structured output prediction problems with latent information, and also efficient to train.

In the case of generative probabilistic models with latent variable, maximum likelihood estimation of a parameter vector $\boldsymbol{\theta}$ involves maximizing the

log-likelihood function

$$\log P_{\boldsymbol{\theta}}(x, y, h),$$

which is usually non-concave. Here x is the observed input features, y is the observed output label, and h is the latent variable. The optimization can be done via the Expectation-Maximization (EM) algorithm [40]. In the Expectation Step (E-Step) we compute the expected log likelihood under the marginal distribution $P_{\boldsymbol{\theta}^{(t)}}(h \mid x, y)$ of the latent variable h, where $\boldsymbol{\theta}^{(t)}$ is the current parameter vector:

$$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) = E_{P_{\boldsymbol{\theta}^{(t)}}(h|x,y)}[\log P_{\boldsymbol{\theta}}(x, y, h)]. \tag{5.1}$$

This fixes the marginal distribution of the latent variables and make the maximization problem simpler. The Maximization Step (M-Step) involves maximizing the function Q for a new parameter $\boldsymbol{\theta}^{(t+1)}$:

$$\boldsymbol{\theta}^{(t+1)} = \operatorname{argmax}_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) \tag{5.2}$$

The EM algorithm alternates between the E-step and the M-step until convergence. Other methods for differentiable optimization such as gradient descent or conjugate gradient can also be applied with suitable initialization. Due to non-concavity of the objective function these algorithms can only converge to a local maximum, and therefore having a good initialization strategy is very important.

How can we construct a general large-margin learning framework that can handle issues such as non-convex objective that comes with the use of latent variables? To tackle this question let us start from the basic joint feature map and prediction. Recall that in structural SVM we are trying to learn a prediction function

$$f_{\boldsymbol{w}}(x) = \operatorname{argmax}_{\hat{y} \in \mathcal{Y}} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x, \hat{y})$$

parameterized by the weight vector $\boldsymbol{w}$ (for simplicity let us restrict our discussion to linear structural SVMs for the moment). Just like the presence of latent variable h in the likelihood function in generative probabilistic models, we can extend our joint feature map to include the influence of latent variables in our prediction function:

$$f_{\boldsymbol{w}}(x) = \text{argmax}_{(\hat{y},\hat{h})\in\mathcal{Y}\times\mathcal{H}}\, \boldsymbol{w}\cdot\boldsymbol{\Phi}(x,\hat{y},\hat{h}). \tag{5.3}$$

By extending the joint feature map from $\boldsymbol{\Phi}(x,y)$ to $\boldsymbol{\Phi}(x,y,h)$, we make sure that latent variables such as word alignments in machine translation or object part positions in object recognition are taken into account when making predictions, even when they are not available in the training set. Notice that we are predicting a pair (y,h) that jointly maximizes the linear scoring function $\boldsymbol{w}\cdot\boldsymbol{\Phi}(x,y,h)$. In a probabilistic model we have the option of integrating out the latent variable h when predicting y as

$$f_{\boldsymbol{\theta}}(x) = \text{argmax}_{\hat{y}\in\mathcal{Y}} \sum_{\hat{h}\in\mathcal{H}} P_{\boldsymbol{\theta}}(x,\hat{y},\hat{h}).$$

In our large-margin framework we replace the sum over latent variables with a point estimate $\hat{h}$ that maximizes the score of the input x and the output y,

$$f_{\boldsymbol{w}}(x) = \text{argmax}_{\hat{y}\in\mathcal{Y}} \max_{\hat{h}\in\mathcal{H}} \boldsymbol{w}\cdot\boldsymbol{\Phi}(x,\hat{y},\hat{h}), \tag{5.4}$$

leading to the above joint prediction rule in Equation (5.3). This point-estimate approach fits much better with the large-margin framework and also avoids the computational issue of integrating out the latent variables.

So far everything looks straightforward as we have done nothing apart from extending the joint feature map and modifying the output type to be a pair (y,h). The interesting part comes when we consider how to formulate this as an optimization problem during training. Unlike the fully observed case where we

know the label y_i for each example x_i, in the latent variable case we only know the label y_i part of the output (y, h) for each example x_i, but have no information about the latent variable h on the training data.

For a start, let us go back to look at the constraints in the standard structural SVM formulation in Optimization Problem 2.7:

$$\begin{aligned}
&\forall \hat{y} \in \mathcal{Y}, \hat{y} \neq y_i, \\
&\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y}) \geq \Delta(y_i, \hat{y}) - \xi_i \\
&\xi_i \geq 0.
\end{aligned}$$

The set of constraints essentially says that the score of the correct output $\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i)$ has to be greater than the score of all other incorrect output $\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y})$, with a margin measured by the loss function $\Delta(y_i, \hat{y})$. We can also rewrite the above set of constraints in a more succinct way as:

$$\xi_i = \max_{\hat{y} \in \mathcal{Y}} [\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y}) + \Delta(y_i, \hat{y})] - \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i)$$

It is easy to see that ξ_i is a convex function of $\boldsymbol{w}$ in this form, since the first term is a maximum over linear functions.

We can preserve the semantics of the above constraints in standard structural SVM in the case with latent variables if we replace the linear scoring function $\boldsymbol{w} \cdot \boldsymbol{\Phi}(x, y)$ in the constraints with the function $\max_{h \in \mathcal{H}} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x, y, h)$:

$$\begin{aligned}
&\forall \hat{y} \in \mathcal{Y}, \hat{y} \neq y_i, \\
&\max_{h \in \mathcal{H}} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i, h) - \max_{\hat{h} \in \mathcal{H}} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y}, \hat{h}) \geq \Delta(y_i, \hat{y}) - \xi_i \\
&\xi_i \geq 0.
\end{aligned}$$

There are two major points to note here. First the new constraints with $\max_{h \in \mathcal{H}} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x, y, h)$ fit well with the prediction rule in Equation (5.4), since

we are using the latent variable h that maximizes the score between the input x and the output y in prediction. Secondly the optimization problem is now non-convex after we put in the standard structural SVM objective due to the term $\max_{h \in \mathcal{H}} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i, h)$, which is a common issue when working with latent variables. This is easy to see if we rewrite the constraints as follows:

$$\xi_i = \max_{(\hat{y}, \hat{h}) \in \mathcal{Y} \times \mathcal{H}} [\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y}, \hat{h}) + \Delta(y_i, \hat{y})] - \max_{h \in \mathcal{H}} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i, h)$$

The slack variable ξ_i is the difference of two convex functions, which is in general not convex. We will discuss how to solve these issues in the next section on training algorithms.

We can now write out our full optimization problem for structural SVM with latent variables, which we call *Latent Structural SVM*:

Optimization Problem 5.1. (LATENT STRUCTURAL SVM)

$$\min_{\boldsymbol{w}, \boldsymbol{\xi}} \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i$$

$$\forall i, \xi_i \geq \max_{(\hat{y}, \hat{h}) \in \mathcal{Y} \times \mathcal{H}} [\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y}, \hat{h}) + \Delta(y_i, \hat{y}, \hat{h})] - \max_{h \in \mathcal{H}} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i, h)$$

Notice we have made the formulation more general by including the latent variable $\hat{h}$ of the alternative output $\hat{y}$ in the loss Δ. This does not change the formulation or the training algorithm in any major manner, but will prove useful in two of the example applications below. It is also easy to see that this formulation reduces to the standard structural SVM if we remove the latent variables.

5.3 Training Algorithm

To solve Optimization Problem 5.1, we are going to exploit a key property that the slack variables ξ_i can be written as the difference of two convex functions. We can rewrite the objective in Optimization Problem 5.1 as:

$$\begin{aligned}
&\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\left(\max_{(\hat{y},\hat{h})\in\mathcal{Y}\times\mathcal{H}}[\boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i,\hat{y},\hat{h}) + \Delta(y_i,\hat{y},\hat{h})] - \max_{h\in\mathcal{H}}\boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i,y_i,h)\right) \\
= &\underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\max_{(\hat{y},\hat{h})\in\mathcal{Y}\times\mathcal{H}}[\boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i,\hat{y},\hat{h}) + \Delta(y_i,\hat{y},\hat{h})]}_{convex} \underbrace{-C\sum_{i=1}^{n}\max_{h\in\mathcal{H}}\boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i,y_i,h)}_{concave}.
\end{aligned}$$

By decomposing the objective as the sum of a convex and a concave part (or difference of two convex functions), we can borrow different tools from the optimization literature [61] to solve the above optimization problem. In particular we are going to employ the Convex-Concave Procedure (CCCP) [156] to solve the above optimization problem. It is a very nice and simple iterative procedure that guarantees to converge to a local minimum or stationary point of the objective $f(\boldsymbol{w}) - g(\boldsymbol{w})$, where f, g are convex functions. The procedure is depicted in Algorithm 5.1.

1: Set $t = 0$ and initialize $\boldsymbol{w}_0$
2: **repeat**
3: Find hyperplane $\boldsymbol{v}_t$ such that $-g(\boldsymbol{w}) \leq -g(\boldsymbol{w}_t) + (\boldsymbol{w} - \boldsymbol{w}_t)\cdot\boldsymbol{v}_t$ for all $\boldsymbol{w}$
4: Solve $\boldsymbol{w}_{t+1} = \operatorname{argmin}_{\boldsymbol{w}} f(\boldsymbol{w}) + \boldsymbol{w}\cdot\boldsymbol{v}_t$
5: Set $t = t + 1$
6: **until** $[f(\boldsymbol{w}_{t-1}) - g(\boldsymbol{w}_{t-1})] - [f(\boldsymbol{w}_t) - g(\boldsymbol{w}_t)] < \epsilon$

Algorithm 5.1: Concave-Convex Procedure (CCCP)

The CCCP algorithm can be viewed as an upper bound minimization procedure. In Line 3 of Algorithm 5.1, we find a linear hyperplane upper bound $\boldsymbol{v}_t$ of the function $-g$ at the current iterate $\boldsymbol{w}_t$. This is always possible because

g is convex, and any subgradient of g at $\boldsymbol{w}_t$ will do. Then we can replace the objective $f - g$ with the convex upper bound:

$$f(\boldsymbol{w}) - g(\boldsymbol{w}) \leq f(\boldsymbol{w}) - g(\boldsymbol{w}_t) + (\boldsymbol{w} - \boldsymbol{w}_t) \cdot \boldsymbol{v}_t. \tag{5.5}$$

This is a global upper bound on the objective and is tight at the current iterate $\boldsymbol{w}_t$. Line 4 of Algorithm 5.1 minimizes this upper bound to obtain the next iterate $\boldsymbol{w}_{t+1}$. The objective $f - g$ is monotonically decreasing for each iteration in the loop and is guaranteed to converge to a local minimum or saddle point.

In terms of the optimization problem for structural SVM with latent variables, the step of computing the upper bound for the concave part in Line 3 involves computing

$$h_i^* = \text{argmax}_{h \in \mathcal{H}} \, \boldsymbol{w}_t \cdot \boldsymbol{\Phi}(x_i, y_i, h) \tag{5.6}$$

for each i. We call this the *latent variable completion* problem. The hyperplane constructed is $\boldsymbol{v}_t = \sum_{i=1}^{n} \boldsymbol{\Phi}(x_i, y_i, h_i^*)$.

Computing the new iterate $\boldsymbol{w}_{t+1}$ in Line 4 involves solving the standard structural SVM optimization problem by completing y_i with the latent variables h_i^* as if they were completely observed:

$$\begin{aligned} \min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \max_{(\hat{y}, \hat{h}) \in \mathcal{Y} \times \mathcal{H}} [\boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, \hat{y}, \hat{h}) + \Delta(y_i, \hat{y}, \hat{h})] \\ - C \sum_{i=1}^{n} \boldsymbol{w} \cdot \boldsymbol{\Phi}(x_i, y_i, h_i^*). \end{aligned} \tag{5.7}$$

This convex optimization problem can be solved using the cutting plane algorithm dicussed in Chapter 3. Thus the CCCP algorithm applied to structural SVM with latent variables gives rise to a very intuitive algorithm that alternates between imputing the latent variables h_i^* that best explain the training pair (x_i, y_i) and solving the structural SVM optimization problem while treating the

latent variables as completely observed. This is similar to the iterative process of Expectation Maximization (EM) [40]. But unlike EM which maximizes the expected log likelihood under the marginal distribution of the latent variables (E-step in Equation (5.1)), we are minimizing the regularized loss against a single latent variable h_i^* that best explains (x_i, y_i). The minimization step of the CCCP algorithm in Equation (5.7) is analogous to the M-step of EM in Equation (5.2).

Summing up, to apply the latent structural SVM learning algorithm for a structured output prediction problem with latent variables, we need to come up with a suitable joint feature map $\mathbf{\Phi}(x, y, h)$ and loss function $\Delta(y, \hat{y}, \hat{h})$ that allows efficient inference in the following problems:

1. Prediction:

$$\text{argmax}_{(\hat{y},\hat{h})\in\mathcal{Y}\times\mathcal{H}}\, \boldsymbol{w} \cdot \mathbf{\Phi}(x_i, \hat{y}, \hat{h})$$

2. Loss-Augmented Inference:

$$\text{argmax}_{(\hat{y},\hat{h})\in\mathcal{Y}\times\mathcal{H}}[\boldsymbol{w} \cdot \mathbf{\Phi}(x_i, \hat{y}, \hat{h}) + \Delta(y_i, \hat{y}, \hat{h})]$$

3. Latent Variable Completion:

$$\text{argmax}_{h\in\mathcal{H}}\, \boldsymbol{w} \cdot \mathbf{\Phi}(x_i, y_i, h)$$

In the following section we will demonstrate how to make these design choices in different applications to enable accurate predictions and efficient training.

5.4 Three Example Applications

5.4.1 Discriminative Motif Finding

Our development of the Latent Structural SVM was motivated by a motif finding problem in yeast DNA through collaboration with computational biologists. Motifs are repeated patterns in DNA sequences that are believed to have biological significance. Our dataset consists of ARSs (autonomously replicating sequences) [22] screened in two yeast species S. kluyveri and S. cerevisiae. Our task is to predict whether a particular sequence is functional (i.e., whether they start the replication process) in S. cerevisiae and to find out the motif responsible. All the native ARSs in S. cerevisiae are labeled as positive, since by definition they are functional. The ones that showed ARS activity in S. kluyveri were then further tested to see whether they contain functional ARS in S. cerevisiae, since they might have lost their function due to sequence divergence of the two species during evolution. They are labeled as positive if functional and negative otherwise. In this problem the latent variable h is the position of the motif in the positive sequences, since current experimental procedures do not have enough resolution to pinpoint their locations. Altogether we have 124 positive examples and 75 negative examples. In addition we have 6460 sequences from the yeast intergenic regions for background model estimation.

Popular methods for motif finding include methods based on EM [11] and Gibbs-sampling [82]. For this particular yeast dataset we believe a discriminative approach, especially one incorporating large-margin separation, is beneficial because of the close relationship and DNA sequence similarity among the different yeast species in the dataset.

Let x_i denote the ith base (A, C, G, or T) in our input sequence x of length n. We use the common position-specific weight matrix plus background model approach in our definition of feature vector:

$$\mathbf{\Phi}(x, y, h) = \begin{cases} \sum_{j=1}^{l} \boldsymbol{\phi}_{PSM}^{(j)}(x_{h+j}) + \sum_{i=1}^{h} \boldsymbol{\phi}_{BG}(x_i) + \sum_{i=h+l+1}^{n} \boldsymbol{\phi}_{BG}(x_i) & \text{[if } y = +1\text{]} \\ \sum_{i=1}^{n} \boldsymbol{\phi}_{BG}(x_i) & \text{[if } y = -1\text{]}, \end{cases}$$

where $\boldsymbol{\phi}_{PSM}^{(j)}$ is the feature count for the jth position of the motif in the position-specific weight matrix, and $\boldsymbol{\phi}_{BG}$ is the feature count for the background model (we use a Markov background model of order 3).

For the positive sequences, we randomly initialized the motif position h uniformly over the whole length of the sequence. We optimized over the zero-one loss Δ for classification and performed a 10-fold cross validation. We make use of the set of 6460 intergenic sequences in training by treating them as negative examples (but they are excluded in the test sets). Instead of penalizing their slack variables by C in the objective we only penalize these examples by $C/50$ to avoid overwhelming the training set with negative examples (with the factor $1/50$ picked by cross-validation). We trained models using regularization constant C from $\{0.1, 1, 10, 100, 1000\}$ times the size of the training set (5992 for each fold), and each model is re-trained 10 times using 10 different random seeds.

As control we ran a Gibbs sampler [97] on the same dataset, with the same set of intergenic sequences for background model estimation. It reports good signals on motif lengths $l = 11$ and $l = 17$, which we compare our algorithm against. To provide a stronger baseline we optimize the classification threshold of the Gibbs sampler on the test set and report the best accuracy over all possible thresholds. Table 5.1 compares the accuracies of the Gibbs sampler and our method averaged across 10 folds. Our algorithm shows a significant improve-

Table 5.1: Classification Error on Yeast DNA (10-fold CV)

	Error rate
Gibbs sampler ($l = 11$)	32.49%
Gibbs sampler ($l = 17$)	31.47%
Latent Structural SVM ($l = 11$)	11.09%
Latent Structural SVM ($l = 17$)	12.00%

ment over the Gibbs sampler (with p-value $< 10^{-4}$ in a paired t-test). As for the issue of local minima, the standard deviations on the classification error over the 10 random seeds, averaged over 10 folds, are 0.0648 for $l = 11$ and 0.0546 for $l = 17$. There are variations in solution quality due to local minima in the objective, but they are relatively mild in this task and can be overcome with a few random restarts.

In this application the latent structural SVM allows us to exploit discriminative information to better detect motif signals compared to traditional unsupervised probabilistic model for motif finding.

5.4.2 Noun Phrase Coreference Resolution

In noun phrase coreference resolution we would like to determine which noun phrases in a text refer to the same real-world entity. In [47] the task is formulated as a correlation clustering problem trained with structural SVMs. In correlation clustering the objective function maximizes the sum of pairwise similarities. However this might not be the most appropriate objective, because in a

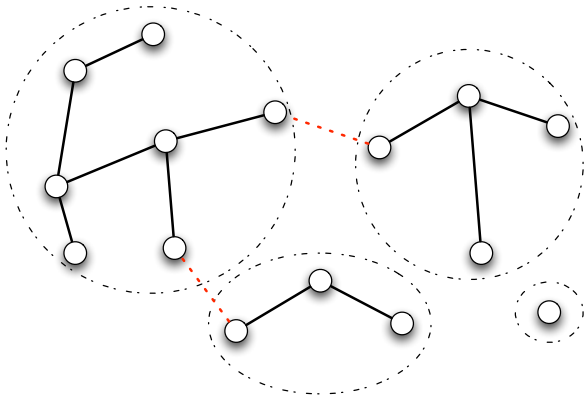


Figure 5.2: The circles are the clusters defined by the label y. The set of solid edges is one spanning forest h that is consistent with y. The dotted edges are examples of incorrect links that will be penalized by the loss function.

cluster of coreferent noun phrases of size k, many of the $O(k^2)$ links contain only very weak signals. For example, it is difficult to determine whether a mention of the name "Tom" at the beginning of a text and a pronoun "he" at the end of the text are coreferent directly without scanning through the whole text.

Following the intuition that humans might determine if two noun phrases are coreferent by reasoning transitively over strong coreference links [98], we model the problem of noun phrase coreference as a single-link agglomerative clustering problem. Each input x contains all n noun phrases in a document, and all the pairwise features $\boldsymbol{x}_{ij}$ between the ith and jth noun phrases. The label y is a partition of the n noun phrases into coreferent clusters. The latent variable h is a spanning forest of "strong" coreference links that is consistent with the clustering y. A spanning forest h is consistent with a clustering y if every cluster in y is a connected component in h (i.e., a tree), and there are no edges in h that connects two distinct clusters in y (Figure 5.2).

To score a clustering y with a latent spanning forest h, we use a linear scoring

model that adds up all the edge scores for edges in h, parameterized by $\boldsymbol{w}$:

$$\boldsymbol{w} \cdot \boldsymbol{\Phi}(x, y, h) = \sum_{(i,j) \in h} \boldsymbol{w} \cdot \boldsymbol{x}_{ij}.$$

To predict a clustering y from an input x (argmax in Equation (5.3)), we can run any Maximum Spanning Tree algorithm such as Kruskal's algorithm on the complete graph of n noun phrases in x, with edge weights defined by $\boldsymbol{w} \cdot \boldsymbol{x}_{ij}$. The output h is a spanning forest instead of a spanning tree because two trees will remain disconnected if all edges connecting the two trees have negative weights. We then output the clustering defined by the forest h as our prediction y.

For the loss function Δ, we would like to pick one that supports efficient computation in the loss-augmented inference, while at the same time penalizing incorrect spanning trees appropriately for our application. We propose the loss function

$$\Delta(y, \hat{y}, \hat{h}) = n(y) - k(y) - \sum_{(i,j) \in \hat{h}} l(y, (i, j)), \tag{5.8}$$

where $n(y)$ and $k(y)$ are the number of vertices and the number of clusters in the correct clustering y. The function $l(y, (i, j))$ returns 1 if i and j are within the same cluster in y, and -1 otherwise. It is easy to see that this loss function is non-negative and zero if and only if the spanning forest $\hat{h}$ defines the same clustering as y. Since this loss function is linearly decomposable into the edges in $\hat{h}$, the loss-augmented inference can also be computed efficiently using Kruskal's algorithm. Similarly the step of completing the latent variable h given a clustering y, which involves computing a highest scoring spanning forest that is consistent with y, can also be done with the same algorithm.

To evaluate our algorithm, we performed experiments on the MUC6 [53] noun phrase coreference dataset. There are 60 documents in the dataset and we

Table 5.2: Clustering Accuracy on MUC6 Data

	MITRE Loss	Pair Loss
SVM-cluster	41.3	2.89
Latent Structural SVM	44.1	2.66
Latent Structural SVM (modified loss, $r = 0.01$)	**35.6**	4.11

use the first 30 for training and the remaining 30 for testing. The pairwise features $\boldsymbol{x}_{ij}$ are the same as those in [98]. The regularization parameter C is picked from 10^{-2} to 10^{6} using a 10-fold cross validation procedure. The spanning forest h for each correct clustering y is initialized by connecting all coreferent noun phrases in chronological order (the order in which they appear in the document), so that initially each tree in the spanning forest is a linear chain.

Table 5.2 shows the result of our algorithm compared to the SVM correlation clustering approach in [47]. We present the results using the same loss functions as in [47]. Pair loss is the proportion of all $O(n^2)$ edges incorrectly classified. MITRE loss is a loss proposed for evaluating noun phrase coreference that is related to the F_1-score [134].

We can see from the first two lines in the table that our method performs well on the Pair loss but worse on the MITRE loss when compared with the SVM correlation clustering approach. Error analysis reveals that our method trained with the loss defined by Equation (5.8) is very conservative when predicting links between noun phrases, having high precision but rather low recall. Therefore we adapt our loss function to make it more suitable for minimizing the MITRE loss. We modified the loss function in Equation (5.8) to penalize

less for adding edges that incorrectly link two distinct clusters, using a penalty $r < 1$ instead of 1 for each incorrect edge added. With the modified loss (with $r = 0.01$ picked via cross-validation) our method performs much better than the SVM correlation clustering approach on the MITRE loss (p-value < 0.03 in a Z-test).

Unlike the SVM correlation clustering approach, where approximate inference is required, our inference procedure involves only simple and efficient maximum spanning tree calculations. For this noun phrase coreference task, the new formulation with latent structural SVM improves both the prediction performance and training efficiency over conventional structural SVMs.

5.4.3 Optimizing Precision@k in Ranking

Our last example application is related to optimizing for precision@k in document retrieval. Precision@k is defined to be the number of relevant documents in the top k positions given by a ranking, divided by k. For each example in the training set, the pattern x is a collection of n documents $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ associated with a query q, and the label $y \in \{-1, 1\}^n$ classifies whether each document in the collection is relevant to the query or not. However for the purpose of evaluating and optimizing for information retrieval performance measures such as precision@k and NDCG@k (normalized discounted cumulative gain), the partial order of the documents given by the label y is insufficient. The label y does not tell us which the top k documents are. To deal with this problem, we can postulate the existence of a latent total order h on all documents related to the query, with h consistent with the partial order given by label y. To be precise, let

h_j be the index of the jth most relevant document, such that $\boldsymbol{x}_{h_j} \geq_{tot} \boldsymbol{x}_{h_{j+1}}$ for j from 1 to $n-1$, where $\geq_{tot}$ is a total order of relevance on the documents $\boldsymbol{x}_i$, and let $>_{tot}$ be its strict version. The label y is consistent with the latent variable h if $y_i > y_j$ implies $\boldsymbol{x}_i >_{tot} \boldsymbol{x}_j$, so that all relevant documents in y comes before the non-relevant documents in the total order h. For optimizing for precision@k in this section, we can restrict h to be first k documents $h_1, \ldots, h_k$.

We use the following construction for the feature vector (in a linear feature space):

$$\boldsymbol{\Phi}(x, y, h) = \frac{1}{k} \sum\nolimits_{j=1}^{k} \boldsymbol{x}_{h_j}.$$

The feature vector only consists of contributions from the top k documents selected by h, when all other documents in the label y are ignored (with the restriction that h has to be consistent with y).

For the loss we use the following precision@k loss:

$$\Delta(y, \hat{y}, \hat{h}) = \min\{1, \frac{n(y)}{k}\} - \frac{1}{k} \sum_{j=1}^{k} [y_{h_j} == 1].$$

This loss function is essentially one minus precision@k, with slight modifications when there are less than k relevant documents in a collection. We replace 1 by $n(y)/k$ so that the loss can be minimized to zero, where $n(y)$ is the total number of relevant documents in y.

Intuitively, with this particular design of the feature vector and the loss function, the algorithm is trying to optimize for the classification accuracy in the region near the top k documents, while ignoring most of the documents in the rest of the feature space (Figure 5.3).

All the inference problems required for this application are efficient to solve. Prediction requires sorting based on the score $\boldsymbol{w} \cdot \boldsymbol{x}_j$ in decreasing order and

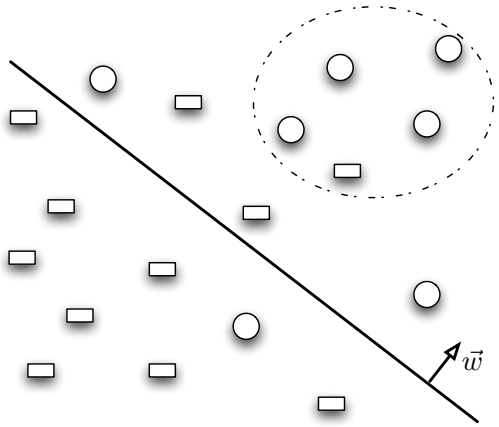


Figure 5.3: Latent Structural SVM tries to optimize for accuracy near the region for the top k documents (circled), when a good general ranking direction $\boldsymbol{w}$ is given

picking the top k. The loss-augmented inference requires sorting based on the score $\boldsymbol{w} \cdot \boldsymbol{x}_j - [y_j == 1]$ and picking the top k for $\hat{h}$. Latent variable completion for y requires a similar sorting procedure on $\boldsymbol{w} \cdot \boldsymbol{x}_j$ and picking the top k, but during sorting the partial order given by the label y has to be respected (so that $\boldsymbol{x}_i$ comes before $\boldsymbol{x}_j$ when either $y_i > y_j$, or $y_i == y_j$ and $\boldsymbol{w} \cdot \boldsymbol{x}_i > \boldsymbol{w} \cdot \boldsymbol{x}_j$).

To evaluate our algorithm, we ran experiments on the OHSUMED tasks of the LETOR 3.0 dataset [86]. We use the per-query-normalized version of the features in all our training and testing below, and employ exactly the same training, test, and validation sets split as given.

For this application it is vital to have a good initialization of the latent varibles h. Simple initialization strategies such as randomly picking k relevant documents indicated by the label y does not work for these datasets with noisy relevance judgements, which usually give the trivial zero vector as solution. Instead we adopt the following initialization strategy. Using the same training and validation sets in each fold, we trained a model optimizing for weighted

Table 5.3: Precision@k on OHSUMED dataset (5-fold CV)

OHSUMED	P@1	P@3	P@5	P@10
Ranking SVM	0.597	0.543	0.532	0.486
ListNet	0.652	0.602	0.550	0.498
Latent Structural SVM	0.680	0.573	0.567	0.494
Initial Weight Vector	0.626	0.557	0.524	0.464

average classification accuracy (weighted by the reciprocal of the number of documents associated by each query). Then for each fold the trained model is used as the initial weight vector to optimize for precision@k.

We can see from Table 5.3 that our latent structural SVM approach performs better than the Ranking SVM [59, 64] on precision@1,3,5,10, one of the stronger baselines in the LETOR 3.0 benchmark. We also essentially tie with ListNet [21], one of the best overall ranking method in the LETOR 3.0 benchmark. As a sanity check, we also report the performance of the initial weight vectors used for initializing the CCCP. The latent structural SVM consistently improves upon these, showing that the good performance is not simply a result of good initialization.

5.5 Conclusions

In this chapter we have presented the Latent Structural SVM algorithm, an extension to strucutral SVM that allows us to incorporate latent variables in learning discriminative models for structured output prediction. This extension greatly increases the application scope of the large margin framework to

different structured output prediction problems. The resulting non-convex objective of this new formulation is optimized efficiently via the Concave-Convex Procedure. Like standard structural SVMs, users can easily adapt latent structural SVMs for different applications by supplying a joint feature map, a loss function, and the corresponding inference procedures. We demonstrated how these design choices can be made with three different example applications and also showed in these improved prediction accuracies when latent variables are incorporated.

5.6 Bibliographical Notes

This work on extending structural SVMs to handle latent variables was first published at [151].

For discriminative modeling with latent variables, there were independent works on extending conditional random fields with hidden variables [140, 54, 101]. These authors have different applications in mind, including using latent variables to model body parts in gesture recognition, to model speech signals, and to model mixtures of part-of-speech tags. Training these models involve optimizing a non-convex log likelihood objective, which can be done either by Expectation Maximization as in [140] or direct gradient descent [54]. However compared with our latent structural SVM model the main computational disadvantage is the computation of the partition function. For example, the work in [101] requires approximation to the feature expectations to make their large scale discriminative training possible.

Close to the time of publication of our work in [151] there were works on

training hidden CRF with the max-margin criterion [45, 141] in the computer vision community. However these works focus on classification problems only and are special cases of our latent structural SVM formulation, and their training algorithms are also derived differently. These formulations of latent SVMs have also been applied recently to tackle problems in natural language processing as well [27].

The Concave-Convex Procedure [156] employed in our work is a general framework for minimizing non-convex functions which falls into the class of DC (Difference of Convex) programming [61]. The work in [121] gave a detailed analysis of the convergence properties of the CCCP algorithm. In recent years there have been numerous applications of the algorithm in machine learning, including training non-convex and transductive SVMs [33], large-margin multiple instance learning [28, 18]. The approach in [120] employs CCCP to handle missing data in SVMs and Gaussian Processes and is closely related to our work. However our approach is non-probabilistic and avoids the computation of partition functions, which is particularly attractive for structured prediction. Very recently the CCCP algorithm has also been applied to obtain tighter non-convex loss bounds on structured learning [24].

CHAPTER 6

TRAINING STRUCTURAL SVMS WITH NONLINEAR KERNELS

In the last few chapters we have been focusing almost exclusively on linear structural SVMs. In this chapter we will study the use of nonlinear kernels in structural SVMs, which greatly improves the flexibility of decision boundaries and feature functions. We are going to focus specifically on the problem of long training time associated with the use of kernels in structural SVMs, and propose two approximation algorithms based on the cutting plane method introduced in Chapter 3. We provide termination iteration bound and also approximation error bounds on the regularized risk. Experimental results show that our algorithms offer one to two order of magnitude improvements in the training time while one of them also offer improved solution sparsity when compared against conventional training method such as the dual decomposition methods used in SVM^{light}.

6.1 Nonlinear Support Vector Machines with Kernels

In Chapter 3 we briefly introduced nonlinear SVMs with kernels, and now we are going to discuss in greater details how to employ kernels to produce nonlinear decision boundaries in SVM. Suppose we are given a training set $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ are the feature vectors and the $y_i \in \{+1, -1\}$ are the labels. Recall the dual optimization problem of binary

classification SVM [cf. Chapter 3],

$$\begin{aligned} \max_{\boldsymbol{\alpha}} & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j \\ & \forall i, 1 \leq i \leq n, 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^{n} y_i \alpha_i = 0. \end{aligned} \tag{6.1}$$

The weight vector can be recovered by the Karush-Kuhn-Tucker conditions by:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i.$$

We mentioned in Chapter 2 that Vapnik and coworkers had the insight that all the distance information between points in the feature space $\mathbb{R}^d$ was expressed through the inner product $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ only, and they can replace this inner product with a kernel function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ that implicitly defines an inner product in a higher dimensional inner product space. This gives us a new dual optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ & \forall i, 1 \leq i \leq n, 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^{n} y_i \alpha_i = 0, \end{aligned} \tag{6.2}$$

The "weight vector" $\boldsymbol{w}$ is defined implicitly in this case as:

$$\boldsymbol{w} \cdot \boldsymbol{z} = \sum_{i=1}^{n} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{z}),$$

where the inner product "$\cdot$" is the inner product in a specially constructed (potentially infinite-dimensional) vector space called the *Reproducing Kernel Hilbert Space* (RKHS). We will discuss this reproducing kernel Hilbert space in more details below. This replacement of the linear inner product with a kernel function is usually referred to as the "kernel trick".

Commonly used kernels include the polynomial kernel (of degree d)

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i \cdot \boldsymbol{x}_j)^d \tag{6.3}$$

and the Gaussian kernel (with width parameter σ)

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{\sigma}). \tag{6.4}$$

These kernels satisfy the positive semidefinite property that for any finite set of inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, the $n \times n$ matrix $\boldsymbol{K}$ formed by $\boldsymbol{K}_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is positive semidefinite. This positive semidefinite property is essential for the kernel K to behave like an inner product in a higher dimensional space, and also for the optimization problem in Equation (6.2) to be convex. The objects in the input space $\mathcal{X}$ are not restricted to be feature vectors in Euclidean spaces, they can be any general objects such as sequences [142, 87, 83], trees [31, 57], patches of images [52]. The similarity/distance measure K on these objects is a valid kernel if it satisfies the positive semidefinite property.

So far we have motivated kernels via replacing the inner products in the dual optimization problem with a more general "inner product"-like kernel function in higher dimensional space. However we could have developed the whole theory for SVMs starting from kernels, with the linear version with weight vector $\boldsymbol{w}$ a finite dimensional special case. Given a valid positive semidefinite kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, we can construct a Hilbert Space $\mathcal{H}$ with all elements $f \in \mathcal{H}$ being functionals of the form $f : \mathcal{X} \to \mathbb{R}$. We can think of elements in $\mathcal{H}$ as scoring functions for input objects in $\mathcal{X}$. A Hilbert space is a complete inner product space. An inner product space is a vector space with an inner product function '$\cdot$' that acts like the dot-product in Euclidean spaces, so that familiar notions such as angles and distances can be applied to abstract objects such as functionals from input space $\mathcal{X}$ to real numbers $\mathbb{R}$. *Complete* means that every

Cauchy sequence has its limit point inside the Hilbert space, using the metric induced by the inner product. Roughly speaking this means any sequence of points in the Hilbert space that are getting closer and closer together (with distance measured by the inner product function) will converge to a point inside the Hilbert space. The Hilbert space has the following *reproducing property*:

$$\forall f \in \mathcal{H}, \forall x \in \mathcal{X}, f(x) = f \cdot_{\mathcal{H}} K(., x),$$

where $\cdot_{\mathcal{H}}$ is the inner product of the Hilbert space $\mathcal{H}$, and $K(., x)$ is the functional $\mathcal{X} \rightarrow \mathbb{R}$ defined by $K(., x)(z) = K(x, z)$. This Hilbert space is therefore usually referred to as *Reproducing Kernel Hilbert Space* (RKHS). Given a positive semidefinite kernel K, the reproducing kernel Hilbert space defined by this kernel is unique (Moore-Aronszajn theorem [8]).

The learning problem for support vector machines can be written as finding a functional f in this Hilbert space that minimizes the following regularized risk:

$$\begin{aligned} \min_{f \in \mathcal{H}, b \in \mathbb{R}} & \frac{1}{2}\|f\|_{\mathcal{H}}^2 + C \sum_{i=1}^{n} \xi_i \\ & s.t.\ y_i[f(x_i) + b] \geq 1 - \xi_i, 1 \leq i \leq n \\ & \xi_i \geq 0, 1 \leq i \leq n \end{aligned}$$

By the *Representer Theorem* [76, 110], the minimizer $f \in \mathcal{H}$ has the following form:

$$f = \sum_{i=1}^{n} \beta_i K(., x_i)$$

for $\beta_i \in \mathbb{R}$. Notice that this corresponds exactly to the expansion of the weight vector $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$ in linear SVMs, where $\beta_i = \alpha_i y_i$ This important result allows us to search for a functional in a potentially infinite-dimensional space $\mathcal{H}$

by solving an optimization problem over a finite set of variables $\beta_1, \ldots, \beta_n$, thus allowing flexible nonlinear decision boundaries.

On the other hand, the new kernel representation of f makes its evaluation expensive. To compute $\boldsymbol{w} \cdot \boldsymbol{z} = (\sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i) \cdot z$ in the linear case, we can first compute $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$ and then compute its inner product with $\boldsymbol{z}$. However we have no such explicit representation for $\boldsymbol{w}$ when nonlinear kernels are used. To classify each input $\boldsymbol{z}$, we need to compute

$$f(\boldsymbol{z}) = \sum_{i=1}^{n} \beta_i K(\boldsymbol{z}, \boldsymbol{x}_i),$$

which takes $O(n)$ evaluations of the kernel function. This is prohibitively expensive for large datasets. Indeed Steinwart [122] has shown that for support vector machine solutions the number of non-zero coefficients β_i is at least $O(n)$. In other words the number of support vectors (and hence the function evaluation cost) grows at least linearly as the training set size.

6.2 Kernels in Structural SVMs

The idea of nonlinear kernels can be quite easily extended to structural SVMs. Instead of directly introducing kernels to replace the joint feature vector $\boldsymbol{\Phi}(x, y)$, it is much more common to introduce nonlinear kernels to replace some of the feature functions in some of the "parts" in the joint input-output structure pair (x, y). For example, let us consider the problem of learning to label Markov Random Fields (MRFs), which is a very general class of structured output learning problems and includes the discriminative training of hidden Markov models as a special case. Let the input x be a graph with input features at its nodes. Let $V(x)$, $E(x)$ be its vertex set and edge set respectively. Denote the set of input

features at node u as x_u, and the labeling at u by y as y_u. If we only use node and edge feature functions for scoring the MRF, then the joint feature vector decomposes naturally as:

$$\Phi(x, y) = \sum_{(u,v) \in E(x)} \boldsymbol{\phi}_{uv}(x_u, x_v, y_u, y_v) + \sum_{u \in V(x)} \boldsymbol{\phi}_u(x_u, y_u). \tag{6.5}$$

We can very easily replace the node feature function $\boldsymbol{\phi}_u$ or the edge feature function $\boldsymbol{\phi}_{uv}$ with nonlinear feature functions in an RKHS for some chosen kernel function K. In this case the feature space for the joint feature map $\boldsymbol{\Phi}(x, y)$ will be a direct sum $\mathbb{R}^d \oplus \mathcal{H}$ between a finite dimensional space $\mathbb{R}^d$ for the linear features and a potentially infinite dimensional RKHS $\mathcal{H}$ for the nonlinear feature functions.

For example, we can keep the edge features $\boldsymbol{\phi}_{uv}$ linear and make the node feature $\boldsymbol{\phi}_u$ a Gaussian kernel. This is similar to the generative Gaussian HMM, and such a model structure is very commonly used in speech recognition. Similarly Taskar et.al [125] replaced the node feature functions $\boldsymbol{\phi}_u$ with polynomial kernels while keeping the edge features $\boldsymbol{\phi}_{uv}$ linear in their handwriting recognition experiments. In our protein alignment problem in Chapter 4 we could also have introduced Gaussian kernels for measuring the similarities between different values of exposed surface area to water, instead of binning these continuous values into discrete bins.

6.3 Cutting Plane Algorithm for Kernel SVMs

In the following sections of this chapter we will investigate how we can adapt the cutting plane algorithm we proposed in Chapter 3 for training nonlinear

structural SVMs with kernels. In this and the following two sections we will focus exclusively on kernels applied to binary classification SVMs, since they are the simplest special case of structural SVMs for illustrating the major ideas without clutter. However the ideas can be applied to general structural SVMs with nonlinear feature functions in a straightforward manner, and we shall have a further discussion on this in Section 6.6.

For simplicity we still use the notation in the linear case $\boldsymbol{w}$ to denote the functional f we want to learn in the RKHS, and we use $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{H}$ to denote the nonlinear map that maps the input features x to the RKHS $\boldsymbol{\phi}(x)$ such that

$$\boldsymbol{\phi}(x) \cdot \boldsymbol{\phi}(z) = K(x, z) \quad \text{for all } x, z \in \mathcal{X}.$$

This map $\boldsymbol{\phi}$ always exists by Mercer's theorem [93]. To simplify notations we use '$\cdot$' to denote the special inner product '$\cdot_{\mathcal{H}}$' in the RKHS for the rest of this chapter.

Nonlinear kernel SVMs are traditionally trained with decomposition algorithms such as Sequential Minimal Optimization (SMO) [102] or working set decomposition [99, 63]. However even with the use of clever heuristics exact training is still slow for large datasets (e.g., training sets with 100K examples). The cutting plane algorithm we introduced in Chapter 3 for training structural SVMs can be quite easily adapted to train nonlinear binary SVMs and nonlinear structural SVMs with kernels, as shown in Algorithm 6.1. This is a special case of the structural SVM formulation, with the joint feature map $\boldsymbol{\Phi}$ being:

$$\boldsymbol{\Phi}(\boldsymbol{x}_i, y_i) = \frac{1}{2} y_i \boldsymbol{\phi}(\boldsymbol{x}_i).$$

The loss function is the zero-one loss ($\Delta(y_i, \bar{y}) = [y_i \neq \bar{y}]$) in this case.

The primal training problem for nonlinear binary classification SVM can be

written as:

Optimization Problem 6.1. (KERNEL SVM FOR BINARY CLASSIFICATION (PRIMAL))

$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$
$$s.t.\ \forall i, y_i(\boldsymbol{w}\cdot\boldsymbol{\phi}(\boldsymbol{x}_i)) \geq 1-\xi_i$$
$$\forall i, \xi_i \geq 0$$

With nonlinear kernels the cutting planes also become functionals in the RKHS, and just like the weight vector they can be written as

$$\boldsymbol{g} = \sum_{j=1}^{k}\beta_j\boldsymbol{\phi}(\boldsymbol{z}_j),$$

where $k \in \mathbb{N}$, $\beta_j \in \mathbb{R}$, and $\boldsymbol{z}_j \in \mathbb{R}^d$. Inner products between cutting planes are computed implicitly via kernel functions:

$$\boldsymbol{g}\cdot\boldsymbol{g}' = (\sum_{j=1}^{k}\beta_j\boldsymbol{\phi}(\boldsymbol{z}_j))\cdot(\sum_{i=1}^{l}\gamma_i\boldsymbol{\phi}(\boldsymbol{u}_i)) = \sum_{j=1}^{k}\sum_{i=1}^{l}\beta_j\gamma_i K(\boldsymbol{z}_j,\boldsymbol{u}_i).$$

The cutting plane algorithm works largely the same once we replace the inner product computations in the constraints and argmax computation with the appropriate kernel function evaluations. Note that the computation of $\bar{y}_i$ at Line 10 is equivalent to the argmax computation $\text{argmax}_{\bar{y}\in\mathcal{Y}}[\Delta(y_i,\bar{y}) + \boldsymbol{w}\cdot\boldsymbol{\Phi}(x_i,\bar{y})]$ for cutting plane generation under the choice of joint feature map and loss function for binary classification above. However the cutting plane generation is very expensive in the case with nonlinear kernels, for the exact same reason of why classification is slow for nonlinear kernel SVM in Section 6.1.

By expanding the weight vector $\boldsymbol{w}$ as a linear combination of cutting planes, we can see that the argmax computation in Line 10 of Algorithm 6.1 involves

1: Input: $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n))$, C, ϵ, $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{x}')$
2: $\boldsymbol{c} \leftarrow \mathbf{0}$, $\boldsymbol{H} \leftarrow \mathbf{0}$, $t \leftarrow 0$
3: **repeat**
4: $\quad \boldsymbol{H} \leftarrow (\boldsymbol{H}_{ij})_{1 \le i,j \le t}$, where $\boldsymbol{H}_{ij} = \boldsymbol{g}^{(i)} \cdot \boldsymbol{g}^{(j)}$
5: $\quad \boldsymbol{\alpha} \leftarrow \operatorname{argmax}_{\boldsymbol{\alpha} \ge 0} \boldsymbol{\alpha}^T \boldsymbol{c} - \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha}$ s.t. $\boldsymbol{\alpha}^T \mathbf{1} \le C$
6: $\quad \xi \leftarrow \frac{1}{C}(\boldsymbol{\alpha}^T \boldsymbol{c} - \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha})$
7: $\quad \boldsymbol{w} \leftarrow \sum_i \alpha_i \boldsymbol{g}^{(i)}$
8: $\quad t \leftarrow t + 1$
9: $\quad$ **for** i=1,...,n **do**
10: $\quad\quad \bar{y}_i \leftarrow \operatorname{sign}(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i) - y_i)$
11: $\quad$ **end for**
12: $\quad \boldsymbol{g}^{(t)} \leftarrow \frac{1}{2n} \sum_{i=1}^{n} (y_i - \bar{y}_i) \boldsymbol{\phi}(\boldsymbol{x}_i)$
13: $\quad c^{(t)} \leftarrow \frac{1}{2n} \sum_{i=1}^{n} |y_i - \bar{y}_i|$
14: **until** $\boldsymbol{w} \cdot \boldsymbol{g}^{(t)} \ge c^{(t)} - \xi - \epsilon$
15: return($\boldsymbol{w}$,ξ)

Algorithm 6.1: Cutting plane algorithm for binary classificiation SVM (dual)

computing:

$$\begin{aligned}
\bar{y} &= \operatorname{sign}(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}) - y) \\
&= \operatorname{sign}(\left(\sum_{i=1}^{t} \alpha_i \boldsymbol{g}^{(i)} \right) \cdot \boldsymbol{\phi}(\boldsymbol{x}) - y) \\
&= \operatorname{sign}(\frac{1}{2n} \sum_{i=1}^{t} \sum_{j=1}^{n} \alpha_i (y_j - \bar{y}_j^{(i)}) K(\boldsymbol{x}_j, \boldsymbol{x}) - y),
\end{aligned}$$

where $\bar{y}^{(i)}$ is the constraint computed via Line 10 at the ith iteration.

The number of non-zero coefficients $|y_j - \bar{y}_j^{(i)}|$ (when the labels y_j and $\bar{y}_j^{(i)}$ disagree) is usually proportional to the number of support vectors, which grows linearly with the training set size n [122]. Thus we need $O(n)$ kernel evaluations when computing the above argmax. Since we need to compute the argmax on n examples to generate one cutting plane, the total complexity for each iteration is $O(n^2)$, which is far too much to handle on any medium to large size datasets.

Input: $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n))$, C, ϵ, $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(x) \cdot \boldsymbol{\phi}(x')$
$\boldsymbol{c} \leftarrow \mathbf{0}$, $\boldsymbol{H} \leftarrow \mathbf{0}$, $t \leftarrow 0$
repeat
  $\boldsymbol{H} \leftarrow (\boldsymbol{H}_{ij})_{1 \le i,j \le t}$, where $\boldsymbol{H}_{ij} = \hat{\boldsymbol{g}}^{(i)} \cdot \hat{\boldsymbol{g}}^{(j)}$
  $\boldsymbol{\alpha} \leftarrow \operatorname{argmax}_{\boldsymbol{\alpha} \ge 0} \boldsymbol{\alpha}^T \boldsymbol{c} - \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha}$ s.t. $\boldsymbol{\alpha}^T \mathbf{1} \le C$
  $\xi \leftarrow \frac{1}{C}(\boldsymbol{\alpha}^T \boldsymbol{c} - \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha})$
  $\boldsymbol{w} \leftarrow \sum_{i=1}^{t} \alpha_i \hat{\boldsymbol{g}}^{(i)}$
  $t \leftarrow t + 1$
  for i=1,...,n **do**
    $\bar{y}_i \leftarrow \operatorname{sign}(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i) - y_i)$
  end for
  $\boldsymbol{g}^{(t)} \leftarrow \frac{1}{2n} \sum_{i=1}^{n} (y_i - \bar{y}_i) \boldsymbol{\phi}(\boldsymbol{x}_i)$
  $c^{(t)} \leftarrow \frac{1}{2n} \sum_{i=1}^{n} |y_i - \bar{y}_i|$
  $I \leftarrow \{i \mid |y_i - \bar{y}_i| > 0\}$
  Sample k indices uniformly from I to form S
  $\hat{\boldsymbol{g}}^{(t)} \leftarrow \frac{|I|}{2nk} \sum_{j=1}^{k} (y_{S_j} - \bar{y}_{S_j}) \boldsymbol{\phi}(\boldsymbol{x}_{S_j})$
until $\boldsymbol{w} \cdot \boldsymbol{g}^{(t)} \ge c^{(t)} - \xi - \epsilon$
return($\boldsymbol{w}$,ξ)

Algorithm 6.2: Approximate cutting plane algorithm for binary classification SVM

How can we reduce the complexity of generating the cutting planes in each iteration? Our main idea is to approximate the dense exact cuts $\boldsymbol{g}^{(t)}$ with $O(n)$ non-zero coefficients for $\boldsymbol{\phi}(\boldsymbol{x}_i)$ in its expansion with sparser cuts with at most k non-zero coefficients for $\boldsymbol{\phi}(\boldsymbol{x}_i)$, where $k \ll n$. To implement the idea we consider below two approaches for carrying out the approximation, one via sampling and one via nonlinear preimage optimization.

6.4 Cut Approximation via Sampling

Our first idea is to approximate the true cut $\boldsymbol{g}^{(t)}$ by a sampled cut $\hat{\boldsymbol{g}}^{(t)}$, where

we sample uniformly k indices from the index set

$$I = \{i \mid |y_i - \bar{y}_i| > 0\}.$$

I is the set of indices that contribute to the non-zero coefficients of $\boldsymbol{\phi}(\boldsymbol{x}_i)$ in the kernel expansion of the cut $\boldsymbol{g}^{(t)}$, and is of size $O(n)$. We replace it with a smaller sample S of size k to construct the approximate cut

$$\hat{\boldsymbol{g}}^{(t)} = \frac{|I|}{2nk} \sum_{j=1}^{k} (y_{S_j} - \bar{y}_{S_j}) \boldsymbol{\phi}(\boldsymbol{x}_{S_j}),$$

and solve the optimization problem with these approximate cuts instead.

Since each cutting plane now contains at most k non-zero indices, computing $\boldsymbol{g}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{z})$ requires at most k kernel evaluations. Thus the computation of $\boldsymbol{w} \cdot \boldsymbol{\phi}(z) = \sum_{i=1}^{t} \alpha_i \boldsymbol{g}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{z})$ requires no more than $O(tk)$ kernel evaluations, where t is the iteration number (or the number of active cuts). Since we need to compute $\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i)$ for the argmax of each example $\boldsymbol{x}_i$, the total number of kernel evaluations is $O(tkn)$ per iteration.

However we can reduce this cost further by storing the values of $\boldsymbol{g}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_j)$ for all previous cuts $1 \leq i \leq t$ and all examples $1 \leq j \leq n$. Let $\boldsymbol{A}$ be a $n \times t$ matrix with $A_{ji} = \boldsymbol{g}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_j)$. Then the value of $\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_j)$ be easily computed via a sum $\sum_{i=1}^{t} \alpha_i A_{ji}$, which is $O(t)$ and overall $O(tn)$ for all examples. Updating the matrix $\boldsymbol{A}$ at each iteration requires $O(kn)$ kernel evaluations. Thus the overall complexity per iteration is $O((t + k)n)$. In the iteration bound (very similar to the one in Chapter 3) below we show that t is upper-bounded by a quantity independent of n, and usually a small sample size $k \ll n$ gives us solutions with performance comparable to the exact solutions. This is a marked improvement over the $O(n^2)$ complexity of a direct application of the cutting plane algorithm.

Before evaluating the quality of approximation through experiments, let us

analzye theoretically the convergence properties and approximation quality of the algorithm.

6.4.1 Convergence and Approximation Bounds

We first prove a bound on the number of cutting planes added before convergence. Notice that the bound is independent of the number of examples n.

Termination

Theorem 6.1. *Let $R = \max_{1 \leq i \leq n} \|\boldsymbol{\phi}(\boldsymbol{x}_i)\|$. For any $\epsilon < 8R^2C^2$, the expected number of cutting planes added before Algorithm 6.2 terminates is no more than*

$$\frac{32CR^2}{\epsilon^2(1 - \exp(\frac{-k\epsilon^2}{16CR^2}))}.$$

Proof. This iteration bound is weaker than the $O(1/\epsilon)$ iteration bound in Chapter 3, since random sampling is used to approximate the cutting planes $\boldsymbol{g}^{(i)}$. This worst case behaviour on convergence could occur if the exact cutting plane $\boldsymbol{g}^{(i)}$ has large violation γ, but the random sampling procedure gives us a poor approximation $\hat{\boldsymbol{g}}^{(i)}$ with violation γ' slightly larger than ϵ (abeit with extremely small probability). A tighter iteration bound might be possible with more careful probabilistic analysis, but below we present a simple argument that gives a bound similar to the one in [132].

Consider the following dummy optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{w},\xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\xi \\ &s.t.\ \forall \boldsymbol{d} \in \mathbb{N}^n, \|\boldsymbol{d}\|_1 = k, \forall r \in \mathbb{N}, r \leq n \\ &\quad \boldsymbol{w} \cdot \frac{r}{nk} \sum_{i=1}^{n} d_i y_i \boldsymbol{\phi}(\boldsymbol{x}_i) \geq \frac{r}{n} - \xi. \end{aligned} \tag{6.6}$$

This is a dummy optimization problem as this is just used as a proof device in a dual lower bound argument, and it is not solved by Algorithm 6.2. Notice each sample approximation in Equation (6.4) is a valid constraint for the above optimization problem. By the termination criterion of Algorithm 6.2 we know that the exact cut is ϵ-violated and satisfies $c^{(t)} - \boldsymbol{w}^{(t)} \cdot \boldsymbol{g}^{(t)} > \xi + \epsilon$. However it is not guaranteed that the sampled cut will be ϵ-violated. Let us call those iterations when the sampled cut is violated by more than $\epsilon/2$ "good" iterations (i.e., $c^{(t)} - \boldsymbol{w}^{(t)} \cdot \hat{\boldsymbol{g}}^{(t)} > \xi + \epsilon/2$) and all other iterations "bad" iterations. By Lemma 3.1 each $\epsilon/2$-violated sampled cut increase the dual objective of the above dummy optimization problem by at least

$$\min\{\frac{C\epsilon}{4}, \frac{\epsilon^2}{32R^2}\}.$$

As we assume $\epsilon/2 < 4R^2C^2$, only the second case applies. Since $\boldsymbol{w} = \boldsymbol{0}, \xi = 1$ is a primal feasible solution to the dummy optimization problem with objective C, the dual objective will exceed C after $32R^2C/\epsilon^2$ "good" iterations. The number of "bad" iterations between successive "good" iterations can be bounded using a geometric distribution argument. Given the precondition $c^{(t)} - \boldsymbol{w}^{(t)} \cdot \boldsymbol{g}^{(t)} > \xi + \epsilon$, if the sampled cut is not $\epsilon/2$-violated (i.e., $c^{(t)} - \boldsymbol{w}^{(t)} \cdot \hat{\boldsymbol{g}}^{(t)} \leq \xi + \epsilon/2$), this implies

$$\boldsymbol{w}^{(t)} \cdot \hat{\boldsymbol{g}}^{(t)} - \boldsymbol{w}^{(t)} \cdot \boldsymbol{g}^{(t)} \geq \frac{\epsilon}{2}.$$

By applying a Hoeffding bound argument similar to Lemma 6.2 below, the

probability of this event can be bounded as follows:

$$P(\boldsymbol{w}^{(t)} \cdot \hat{\boldsymbol{g}}^{(t)} - \boldsymbol{w}^{(t)} \cdot \boldsymbol{g}^{(t)} \geq \frac{\epsilon}{2}) < \exp\left(\frac{-k\epsilon^2}{16CR^2}\right)$$

This implies the probability of having a "bad" iteration is no more than

$$P(c^{(t)} - \boldsymbol{w}^{(t)} \cdot \hat{\boldsymbol{g}}^{(t)} \leq \xi + \epsilon/2 \mid c^{(t)} - \boldsymbol{w}^{(t)} \cdot \boldsymbol{g}^{(t)} > \xi + \epsilon) < \exp\left(\frac{-k\epsilon^2}{16CR^2}\right).$$

Thus in expectation there are $1/(1 - \exp(\frac{-k\epsilon^2}{16CR^2}))$ "bad" iterations between successive "good" iterations. Therefore the expected total number of iterations for termination is no more than

$$\frac{32CR^2}{\epsilon^2(1 - \exp(\frac{-k\epsilon^2}{16CR^2}))}.$$

□

Approximation Error Bounds

After proving termination and bounding the number of cutting planes required, we turn our attention to the accuracy of the solutions. Specifically we will characterize the difference between the regularized risk of the exact solution and our approximate solutions. The main idea used in the proof is: if the error introduced by each approximate cut is small with high probability, then the difference between the exact and approximate solutions will also be small with high probability. Bounding the difference between the exact cut and the sampled cut can be done with Hoeffding's inequality.

Let us start the proofs by defining some notation. Let $f(\boldsymbol{w}) = \max_{1 \leq t \leq T} \left(c^{(t)} - \boldsymbol{w} \cdot \boldsymbol{g}^{(t)}\right)$ be an exact cutting plane model of the empirical risk, and

let $\tilde{f}(\boldsymbol{w}) = \max_{1 \le t \le T} \left(c^{(t)} - \boldsymbol{w} \cdot \hat{\boldsymbol{g}}^{(t)}\right)$ be an approximate cutting plane model, with $(c^{(t)}, \hat{\boldsymbol{g}}^{(t)})$ being the approximate cutting planes. We have the following lemma:

Lemma 6.1. *Let a fixed $\boldsymbol{v}$ in the RKHS $\mathcal{H}$ be given. Suppose for some $\gamma > 0$ each of the cutting plane and its approximate counterpart satisfy*

$$P\left((c^{(t)} - \boldsymbol{v} \cdot \hat{\boldsymbol{g}}^{(t)}) - (c^{(t)} - \boldsymbol{v} \cdot \boldsymbol{g}^{(t)}) \ge \gamma\right) < p_\gamma,$$

for $t = 1 \dots T$. Then $\tilde{f}(\boldsymbol{v}) < f(\boldsymbol{v}) + \gamma$ with probability at least $1 - Tp_\gamma$.

Proof. By union bound we know that $(c^{(t)} - \boldsymbol{v} \cdot \hat{\boldsymbol{g}}^{(t)}) - (c^{(t)} - \boldsymbol{v} \cdot \boldsymbol{g}^{(t)}) < \gamma$ for $1 \le t \le T$ occurs with probability at least $1 - Tp_\gamma$. The following chain of implications holds:

$$\begin{aligned}
&\bigwedge_{t=1}^{T} \left((c^{(t)} - \boldsymbol{v} \cdot \hat{\boldsymbol{g}}^{(t)}) - (c^{(t)} - \boldsymbol{v} \cdot \boldsymbol{g}^{(t)}) < \gamma\right) \\
\Rightarrow\ &\max_{1 \le t \le T} \left((c^{(t)} - \boldsymbol{v} \cdot \hat{\boldsymbol{g}}^{(t)}) - (c^{(t)} - \boldsymbol{v} \cdot \boldsymbol{g}^{(t)})\right) < \gamma \\
\Rightarrow\ &\max_{1 \le t \le T} (c^{(t)} - \boldsymbol{v} \cdot \hat{\boldsymbol{g}}^{(t)}) - \max_{1 \le t \le T} (c^{(t)} - \boldsymbol{v} \cdot \boldsymbol{g}^{(t)}) < \gamma
\end{aligned}$$

Hence $\tilde{f}(\boldsymbol{v}) < f(\boldsymbol{v}) + \gamma$ with probability at least $1 - Tp_\gamma$. □

The lemma shows that the approximate cutting plane model does not overestimate the loss by more than a certain amount with high probability. Now we are going to use this lemma to analyze the sampling approximation algorithm Algorithm 6.2. We can bound the difference between the exact cutting planes and the approximate cutting planes using Hoeffding's inequality in the following lemma:

Lemma 6.2. *Let a fixed $\boldsymbol{v} \in \mathcal{H}, \|\boldsymbol{v}\| \le \sqrt{2C}$ be given, and let the exact cutting planes $(c^{(t)}, \boldsymbol{g}^{(t)})$ and approximate cutting planes $(c^{(t)}, \hat{\boldsymbol{g}}^{(t)})$ be defined as in Algorithm 6.2. We*

have for each $t = 1 \dots T$,

$$P\big((c^{(t)} - \boldsymbol{v} \cdot \hat{\boldsymbol{g}}^{(t)}) - (c^{(t)} - \boldsymbol{v} \cdot \boldsymbol{g}^{(t)}) \geq \gamma\big) < \exp\left(\frac{-k\gamma^2}{4CR^2\eta^2}\right)$$

where $\eta = |I|/n$, I being the index set at the t-th iteration.

Proof. Define $Z_j = \boldsymbol{v} \cdot [\frac{1}{2}(y_{S_j} - \bar{y}_{S_j})\boldsymbol{\phi}(x_{S_j})]$. We have

$$\begin{aligned}
\frac{1}{k}\sum_{j=1}^{k} Z_j &= \boldsymbol{v} \cdot \frac{1}{k}\sum_{j=1}^{k}[\frac{1}{2}(y_{S_j} - \bar{y}_{S_j})\boldsymbol{\phi}(x_{S_j})] \\
&= \frac{n}{|I|}\frac{|I|}{2kn}\boldsymbol{v} \cdot \sum_{j=1}^{k}(y_{S_j} - \bar{y}_{S_j})\boldsymbol{\phi}(x_{S_j}) \\
&= \frac{1}{\eta}\boldsymbol{v} \cdot \hat{\boldsymbol{g}}^{(t)}
\end{aligned}$$

Since S_j are sampled uniformly from the index set I, Z_j's are independent with with expectation:

$$\begin{aligned}
E(Z_j) = & -\frac{1}{|I|}\sum_{i \in I} \boldsymbol{v} \cdot [\frac{1}{2}(y_i - \bar{y}_i)\boldsymbol{\phi}(x_i)] \\
&= \frac{n}{|I|}\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{v} \cdot [\frac{1}{2}(y_i - \bar{y}_i)\boldsymbol{\phi}(x_i)] \\
&= \frac{1}{\eta}\boldsymbol{v} \cdot \boldsymbol{g}^{(t)}
\end{aligned}$$

Each Z_j is also bounded between $[-\sqrt{2C}R, \sqrt{2C}R]$. Apply Hoeffding's inequality and we have

$$P(\frac{1}{\eta}\boldsymbol{v} \cdot \boldsymbol{g}^{(t)} - \frac{1}{\eta}\boldsymbol{v} \cdot \hat{\boldsymbol{g}}^{(t)} \geq \gamma) \leq \exp\left(\frac{-2k\gamma^2}{(2\sqrt{2C}R)^2}\right)$$

$$P(\frac{1}{\eta}(c^{(t)} - \boldsymbol{v} \cdot \hat{\boldsymbol{g}}^{(t)}) - \frac{1}{\eta}(c^{(t)} - \boldsymbol{v} \cdot \boldsymbol{g}^{(t)}) \geq \gamma) \leq \exp\left(\frac{-k\gamma^2}{(4CR^2}\right)$$

Put $\gamma = \gamma/\eta$ and we obtain the result. □

Now we are ready to prove our main theorem relating the regularized risk of the optimal solution to our approximate solution. Let $\boldsymbol{v}^*$ be the optimal solution to Optimization Problem 6.1. We have the following theorem:

Theorem 6.2. *Suppose Algorithm 6.2 terminates in T iterations and return $\boldsymbol{w}^*$ as solution. Then with probability at least $1-\delta$,*

$$\frac{1}{2}\|\boldsymbol{w}^*\|^2+CL(\boldsymbol{w}^*)\leq\frac{1}{2}\|\boldsymbol{v}^*\|^2+CL(\boldsymbol{v}^*)+C\left(\epsilon+\sqrt{\frac{4CR^2}{k}\log\frac{T}{\delta}}\right),$$

where $L(\boldsymbol{w})$ is the margin loss $\frac{1}{n}\sum_{i=1}^{n}\xi_i$ as in Optimization Problem 6.1.

Proof. With the exact cutting planes $(c^{(t)},\boldsymbol{g}^{(t)})$ and approximate cutting planes $(c^{(t)},\hat{\boldsymbol{g}}^{(t)})$ as defined in Lemma 6.2, we apply Lemma 6.1. Put $\boldsymbol{v}=\boldsymbol{v}^*$, and $p_\gamma=\exp(-k\gamma^2/4CR^2)$ (we omit η since it is bounded above by 1), we obtain $\tilde{f}(\boldsymbol{v}^*)<f(\boldsymbol{v}^*)+\gamma$ with probability at least $1-T\exp(-k\gamma^2/4CR^2)$. Inverting the statement and we have with probability at least $1-\delta$:

$$\tilde{f}(\boldsymbol{v}^*)<f(\boldsymbol{v}^*)+\sqrt{\frac{4CR^2}{k}\log\frac{T}{\delta}}$$

Since $\boldsymbol{w}^*$ is the optimal solution of $\min_{\boldsymbol{w}}\frac{1}{2}\|\boldsymbol{w}\|^2+C\tilde{f}(\boldsymbol{w})$ at the Tth iteration, we have the following:

$$\begin{aligned}
&\frac{1}{2}\|\boldsymbol{w}^*\|^2+C\tilde{f}(\boldsymbol{w}^*)\leq\ \frac{1}{2}\|\boldsymbol{v}^*\|^2+C\tilde{f}(\boldsymbol{v}^*)\\
<\ &\frac{1}{2}\|\boldsymbol{v}^*\|^2+Cf(\boldsymbol{v}^*)+C\sqrt{\frac{4CR^2}{k}\log\frac{T}{\delta}}\ \text{(with probability } 1-\delta)\\
\leq\ &\frac{1}{2}\|\boldsymbol{v}^*\|^2+CL(\boldsymbol{v}^*)+C\sqrt{\frac{4CR^2}{k}\log\frac{T}{\delta}}\ \text{(subgradient property)}
\end{aligned}\tag{6.7}$$

The last line makes use of the subgradient property that $f(\boldsymbol{w})\leq L(\boldsymbol{w})$ for any exact cutting plane model f of a convex loss function L. Since we are using the exact cutting plane as the condition for exiting the while loop, so we must have

at termination:

$$
\begin{aligned}
c^{(T)} - \boldsymbol{w}^* \cdot \boldsymbol{g}^{(T)} &\leq \xi + \epsilon \\
\frac{1}{n}\sum_{i=1}^{n} \Delta(y_i, \hat{y}_i) - \frac{1}{n}\boldsymbol{w}^* \cdot \sum_{i=1}^{n} [\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \hat{y}_i)] &\leq \tilde{f}(\boldsymbol{w}^*) + \epsilon \\
\max_{(\hat{y}_1, \ldots, \hat{y}_n) \in \mathcal{Y}^n} \frac{1}{n}\sum_{i=1}^{n} [\Delta(y_i, \hat{y}_i) - \boldsymbol{w}^* \cdot [\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \hat{y}_i)]] &\leq \tilde{f}(\boldsymbol{w}^*) + \epsilon \\
L(\boldsymbol{w}^*) &\leq \tilde{f}(\boldsymbol{w}^*) + \epsilon
\end{aligned}
$$

Therefore we have:

$$
\begin{aligned}
& \frac{1}{2}\|\boldsymbol{w}^*\|^2 + CL(\boldsymbol{w}^*) \\
\leq\ & \frac{1}{2}\|\boldsymbol{w}^*\|^2 + C(\tilde{f}(\boldsymbol{w}^*) + \epsilon) \\
\leq\ & \frac{1}{2}\|\boldsymbol{v}^*\|^2 + CL(\boldsymbol{v}^*) + C\left(\epsilon + \sqrt{\frac{4CR^2}{k}\log\frac{T}{\delta}}\right)
\end{aligned}
$$

with probability at least $1 - \delta$. □

Compared to the exact cutting plane algorithm there is an extra error term $\sqrt{\frac{4CR^2}{k}\log\frac{T}{\delta}}$, and Figure 6.1 shows how this error term arise from the use of inexact cutting planes. The theorem shows that as far as obtaining a finite precision solution to the regularized risk minimization problem is concerned, it is sufficient to use sampled cuts with sufficiently large sample size k to match the desired accuracy ϵ of the solution. We will see in the experiment section that fairly small values of k work well in practice.

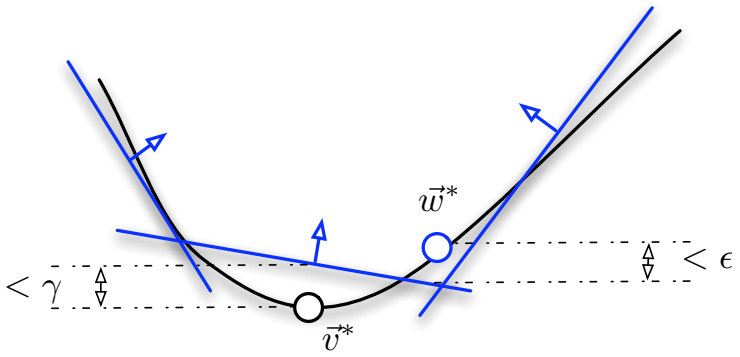


Figure 6.1: Proof Idea for Theorem 6.2: the objective difference between $\boldsymbol{w}^*$ and $\boldsymbol{v}^*$ is bounded by $\epsilon + \gamma$. ϵ is the usual stopping criterion while γ is the error introduced by the use of approximate cutting planes, and is bounded by $\sqrt{\frac{4CR^2}{k} \log \frac{T}{\delta}}$.

6.4.2 Experiments

Experiment Setup

We implemented Algorithm 6.2 and evaluated it on the task of binary classification with kernels. We choose this task for evaluation because binary classification with kernels is a well-studied problem, and there are stable SVM solvers that are suitable for comparisons. Moreover, scaling up SVM with kernels to large datasets is an interesting research problem on its own [32].

In binary classification the loss function Δ is just the zero-one loss. The feature map $\boldsymbol{\Phi}$ is defined by $\boldsymbol{\Phi}(x, y) = (1/2)y\boldsymbol{\phi}(x)$, where $y \in \{1, -1\}$ and $\boldsymbol{\phi}$ is the nonlinear feature map induced from a Mercer kernel (such as the commonly used polynomial kernels and Gaussian kernels).

We implemented the algorithms in C, using Mosek as the quadratic program solver and the SFMT implementation [108] of Mersenne Twister as the ran-

dom number generator. The experiments were run on machines with Opteron 2.0GHz CPUs with 2Gb of memory (with the exception of the control experiments with incomplete Cholesky factorization, which we ran on machines with 4Gb of memory).

For all the experiments below we fix the precision parameter ϵ at 0.001. We remove cuts that are inactive for 20 iterations. For each combination of parameters we ran the experiment for 3 runs using different random seeds, and report the average result in the plots and tables below. In Section 6.4.2 we also investigate the stability of the algorithms by reporting the standard deviation of the results.

In the experiments below we test our algorithms on three different datasets: Checkers, Adult, and Covertype. Checkers is a synthetic dataset with 1 million training points, with classes alternating on a 4x4 checkerboard. We generated the data using the SimpleSVM toolbox [136], with noise level parameter sigma set to 0.02. The kernel width for the Gaussian kernel used for the Checkers dataset was determined by cross validation on a small subsample of 10000 examples. Adult is a medium-sized dataset with 32562 examples, with a sample of 22697 examples taken as training set. The Gaussian kernel width is taken from [102]. Covertype is a dataset with 522910 training points, the kernel width of the Gaussian kernel we use below is obtained from the study [32].

Scaling with Training Set Size

Our first set of experiments is about how the two algorithms scale with training set size. We perform the experiments on the two large datasets Checkers and

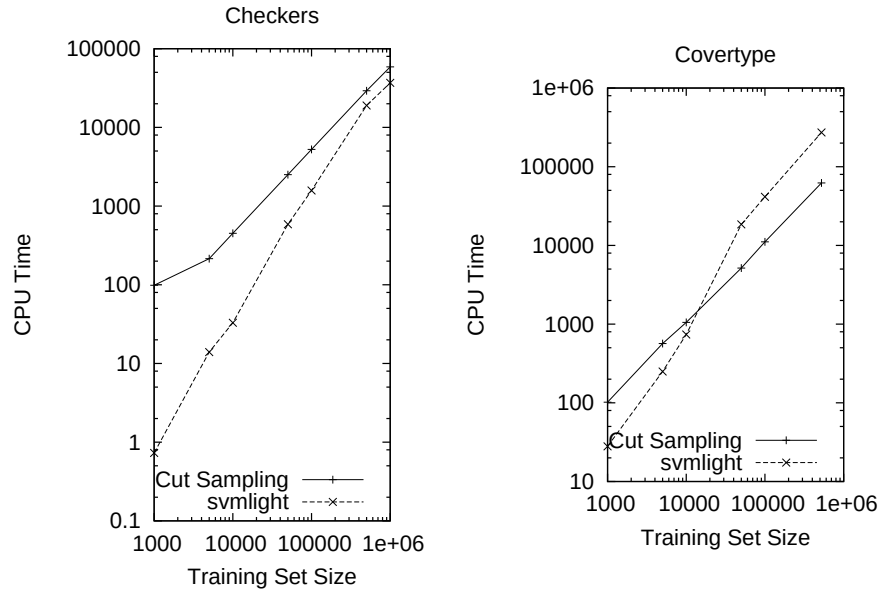


Figure 6.2: CPU Time Against Training Set Size

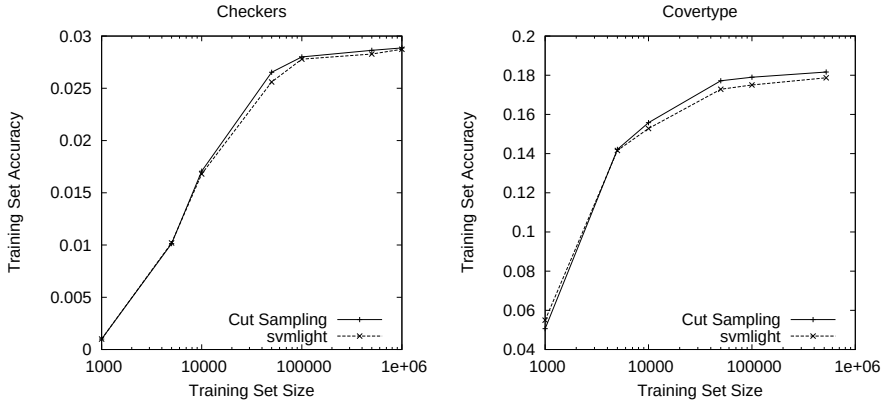


Figure 6.3: Training Set Error Against Training Set Size

Covertype. We pick C to be 1 multiplied by the training set size, since that is the largest value of C we could get SVM^{light} to train within 5 days. We fix the sample size k at 400. We train SVM models on subsets of the full training sets of various sizes to evaluate scaling.

Figure 6.2 shows the CPU time required to train SVMs on training sets of different sizes on the Checkers and Covertype dataset. We can observe that

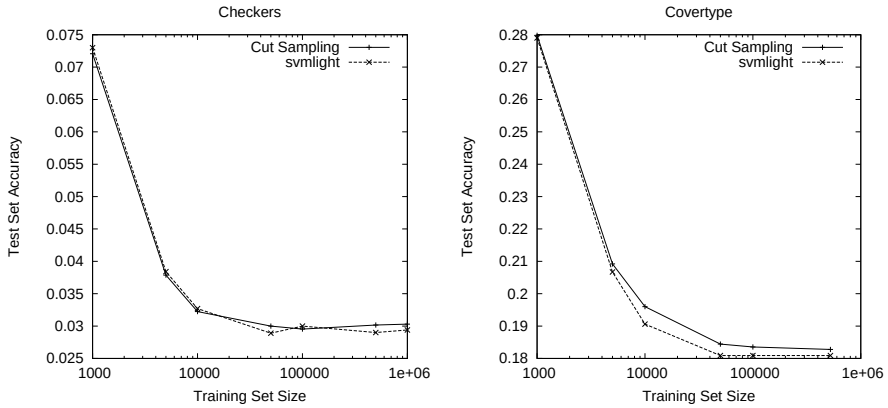


Figure 6.4: Test Set Error Against Training Set Size

the sampling algorithm scales roughly linearly in the log-log plot. This confirms the scaling behaviour we expect from the complexity of each iteration. SVM^{light} shows superlinear scaling on both of these datasets.

Figures 6.3 and 6.4 show the training and test set errors of the algorithms. We can see that the training set accuracies decrease as we increase the training set size while the test set accuracies increase as we increase the training set size. In general SVM^{light} has lower training and test set errors, but the results of the sampling algorithm is also very close. Both the training and test set errors lie within a very narrow band, and they are never more than 0.5 percentage point apart even in the worst case.

Effect of Different Sample Sizes

The next set of experiments is about the effect of the sample size k on training time and solution quality. We investigate the effect of sample size using the Adult dataset, since on this dataset it is easier to collect more data points for different sample sizes. We use sample sizes k from $\{100, 400, 1000, 4000, 10000\}$

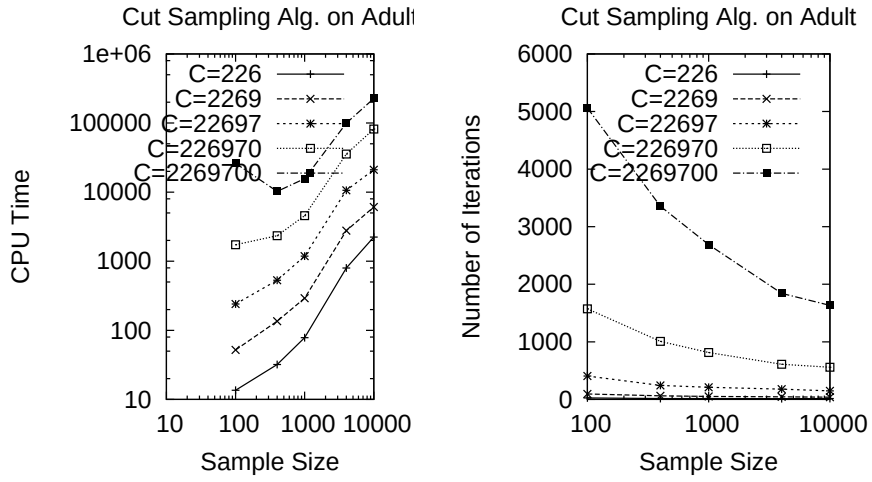


Figure 6.5: CPU Time(Left) and Number of Iteration (Right) Against Sample Size

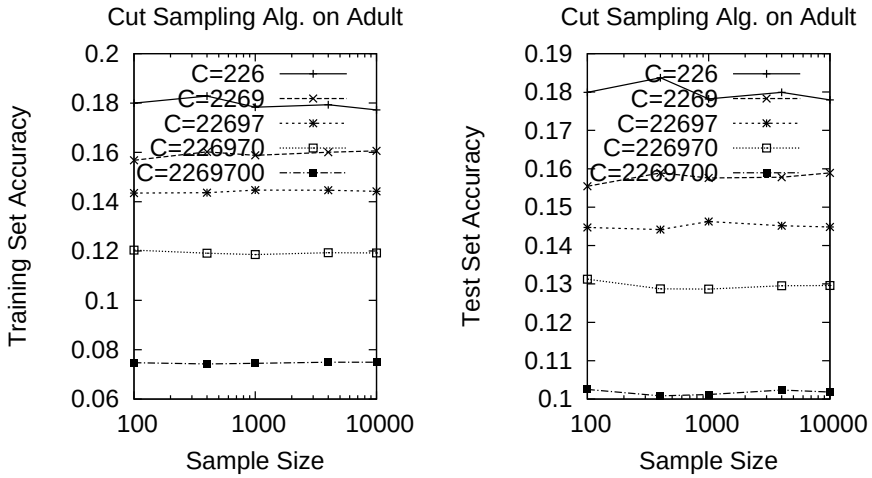


Figure 6.6: Training Set Error(Left) and Test Set Error(Right) Against Sample Size

and C from $\{0.01, 0.1, 1, 10, 100\}$ multiplied by the training set size 22697.

The right of Figure 6.5 shows that the number of iterations required generally decreases with increasing sample size. However the decrease in the number of iterations to convergence does not result in overall savings in time due to the extra cost involved in each iteration with larger sample sizes. This can be ob-

served from the CPU Time plots in the left of Figure 6.5. What is most interesting is the stability of training and test set errors with respect to changes to sample size, as shown in Figure 6.6. Except for very small sample sizes like 100 or small values of C like 0.01 the sets of curves are essentially flat.

Quality of Solutions

Table 6.1 shows a comparison of the two algorithms against two conventional training methods, namely SVM^{light} and a sampling-based method that uses Cholesky decomposition as described below. For each dataset we train different models using values of $C \in \{0.01, 0.1, 1, 10, 100\}$, multipled by the size of the training set. We used the results of SVM^{light} as a yardstick to compare against, and report the value of C for which the test performance is optimal for SVM^{light}. For the larger datasets Checkers and Covertype, SVM^{light} terminated early due to slow progress for $C \geq 10$, so for those two datasets we use $C = 1$.

First of all, we notice from Table 6.1 that all the solutions have training and test set error rates very close to the solutions produced by SVM^{light}. The sampling approximation algorithm has error rates usually within 0.2 above the SVM^{light} solutions. The error rates also have very small standard deviation, on the order of 0.1, which is the same as our tolerance parameter ϵ.

We also provide control experiments with Cholesky decomposition method, where we subsample a set of points from the training set, and then compute the projection of all the points in the training set onto the subspace spanned by these examples. Then we train a linear SVM using $\mathrm{SVM}^{\mathrm{perf}}$[67] (with options `--t 2 -w 3 --b 0`) on the whole training set. Our implementation involves storing

Table 6.1: Runtime and training/test error of sampling algorithms compared to SVM^{light} and Cholesky.

Algorithm	Checkers (N=1000000, C=1, σ^2=0.05) Err(Train)	Err(Test)	CPU sec
Sampling ($r=400$)	2.90(0.01)	2.99(0.01)	60101(1042)
Sampling ($r=1000$)	2.89(0.00)	2.95(0.05)	151759(3334)
Cholesky(250)	3.11	3.32	2447
Cholesky(500)	N/A	N/A	N/A
Cholesky(5000)	N/A	N/A	N/A
Cholesky(10000)	N/A	N/A	N/A
SVM^{light}	2.87	2.94	36533

Algorithm	Adult (N=22697, C=100, σ^2=20) Err(Train)	Err(Test)	CPU sec
Sampling ($r=400$)	7.43(0.09)	10.19(0.15)	11136(640)
Sampling ($r=1000$)	7.44(0.01)	10.19(0.15)	15191(612)
Cholesky(250)	15.10	15.09	604
Cholesky(500)	14.60	14.50	537
Cholesky(5000)	10.89	12.43	3386
Cholesky(10000)	9.12	11.41	8836
SVM^{light}	7.52	10.37	11630

Algorithm	Covertype (N=522910, C=1, σ^2=1.7) Err(Train)	Err(Test)	CPU sec
Sampling ($r=400$)	18.16(0.03)	18.28(0.02)	62184(485)
Sampling ($r=1000$)	17.96(0.03)	18.22(0.02)	155196(6649)
Cholesky(250)	21.73	21.85	1937
Cholesky(500)	20.29	20.35	3093
Cholesky(5000)	N/A	N/A	N/A
Cholesky(10000)	N/A	N/A	N/A
SVM^{light}	17.87	18.09	273021

all the projected training vectors, and this consumes a lot of memory, especially for large datasets like Checkers and Covertype. We can only do 250 and 500 basis functions on those datasets respectively without running out of memory on a 4Gb machine, and on the Adult dataset we can only do up to 10000 basis functions. An alternative implementation with smaller storage requirement would

involve recomputing the projected training vector when needed, but this would become prohibitively expensive.

We observe that the Cholesky decomposition is generally faster than all the other methods, but its accuracy is usually substantially below that of SVM^{light} and our sampling algorithms. Moreover, unlike our algorithms, the accuracy of the Cholesky method depends crucially on the number of basis functions, which is difficult to pick in advance. The accuracies of our sampling algorithms are more stable with respect to the choice of sample size, where decreasing the sample size ususally results in more iterations to converge without much loss in accuracy of the solutions.

6.5 Cut Approximation via Preimage Optimization

In the last section we show that we can reduce the time complexity of kernel SVMs by approximating the dense exact cutting planes with sampling approximation. The approximation algorithm shows good empirical performance, getting very close to the test accuracy of the exact solution while using only relatively small sample size such as 400 or 1000. However, there are still rooms for improvement for the above algorithm. First of all, although the uniform sampling strategy allows us to obtain approximation bounds on the regularized risk, it does not focus on creating a sparse expansion for the weight vector $\boldsymbol{w} = \sum_t \alpha_t \boldsymbol{g}^{(t)}$. Therefore while training is relatively fast since we store all the inner products between the cuts and the examples in a matrix $\boldsymbol{A}$, classification during test time over unseen examples can still be slow because the weight vector can contain $O(Tk)$ basis functions $\boldsymbol{\phi}(\boldsymbol{x}_i)$. For example, the number of basis

functions in the solution for the Adult dataset is very close to the exact solution.

To directly optimize for the sparsity of our solution, we need to be a lot more careful when adding basis functions $\phi(\boldsymbol{x}_i)$ to the cut by making its coefficients non-zero. Methods such as Incomplete Cholesky Decomposition [46] and the Nyström Method [145] choose a subspace in the RKHS $\mathcal{H}$ spanned by a finite basis, and then solve the SVM optimization problem in this subspace. These low rank matrix approximation methods have direct control over the size of set of basis functions, and therefore speed up both training and testing. However, their basis choice are usually done through random sampling and do not take the label information y into account (with the exception of [9]).

In the last section on sampling approximation we have demonstrated through theoretical bounds and empirical results that we can obtain a good solution to the kernel SVM problem by approximating each cutting plane well. In this section we take the idea one step further by combining it with low-rank subspace methods, by searching for a low-rank subspace (or a set of basis functions) that approximates the set of cutting planes well. We can also prove theoretical bounds on the regularized risk objective, and experimental results show that this method produces even sparser and more accurate solutions compared to the sampling approximation. Since our approach is closely related to basis pursuit algorithms [26, 135], we call it the *Cutting Plane Basis Pursuit* (CPSP) algorithm.

Input: $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n))$, C, ϵ, k_{max}, $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$
$\boldsymbol{c} \leftarrow \mathbf{0}$, $\boldsymbol{H} \leftarrow \mathbf{0}$, $\boldsymbol{B} \leftarrow \emptyset$, $t \leftarrow 0$
repeat
  $\boldsymbol{H} \leftarrow (\boldsymbol{H}_{ij})_{1 \leq i,j \leq t}$, where $\boldsymbol{H}_{ij} = \hat{\boldsymbol{g}}^{(i)} \cdot \hat{\boldsymbol{g}}^{(j)}$
  $\boldsymbol{\alpha} \leftarrow \operatorname{argmax}_{\boldsymbol{\alpha} \geq 0} \boldsymbol{\alpha}^T \boldsymbol{c} - \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha}$ s.t. $\boldsymbol{\alpha}^T \mathbf{1} \leq C$
  $\xi \leftarrow \frac{1}{C}(\boldsymbol{\alpha}^T \boldsymbol{c} - \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha})$
  $\boldsymbol{w} \leftarrow \sum_{i=1}^{t} \alpha_i \hat{\boldsymbol{g}}^{(i)}$
  $t \leftarrow t + 1$
  for i=1,...,n **do**
    $\bar{y}_i \leftarrow \operatorname{sign}(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i) - y_i)$
  end for
  $\boldsymbol{g}^{(t)} \leftarrow \frac{1}{2n} \sum_{i=1}^{n} (y_i - \bar{y}_i) \boldsymbol{\phi}(\boldsymbol{x}_i)$
  $c^{(t)} \leftarrow \frac{1}{2n} \sum_{i=1}^{n} |y_i - \bar{y}_i|$
  if $|\boldsymbol{B}| < k_{max}$ **then** $\boldsymbol{B} \leftarrow extend\_basis(\boldsymbol{B}, \boldsymbol{g}^{(t)})$
  for i=1,...,t **do**
    $\hat{\boldsymbol{g}}^{(i)} \leftarrow project(\boldsymbol{g}^{(i)}, \boldsymbol{B})$
  end for
until $\boldsymbol{w} \cdot \boldsymbol{g}^{(t)} \geq c^{(t)} - \xi - \epsilon$
return($\boldsymbol{w}$,ξ)

Algorithm 6.3: Cutting Plane Subspace Pursuit

6.5.1 The Basis

In most works on speeding up the training of kernel SVM in the literature, the basis set $\boldsymbol{B}$ is chosen from the set $\boldsymbol{\phi}(\boldsymbol{x}_i)$ at each example point $\boldsymbol{x}_i$, i.e., $\boldsymbol{B} \subseteq \{\boldsymbol{\phi}(\boldsymbol{x}_i)\}_{i=1}^{n}$ [135, 74]. For commonly used kernel functions such as the Gaussian kernel and the polynomial kernels (Equations (6.3) and (6.4)), the input space is a d-dimensional real vector space. In these cases we can very easily extend the candidates for the basis set $\boldsymbol{B}$ from training examples $\boldsymbol{\phi}(\boldsymbol{x}_i)$ to $\boldsymbol{\phi}(\boldsymbol{b})$ for any $\boldsymbol{b} \in \mathbb{R}^d$. The vector $\boldsymbol{b}$ does not have to be equal to any of the examples $\boldsymbol{x}_i$, and we will see shortly that this relaxation allows us to reduce the approximation error and improve the sparsity.

The whole cutting plane basis pursuit algorithm is illustrated in Algorithm 6.3. The algorithm follows the same template as in most cutting plane algorithms, but it contains several differences when compared to the sampling approximation in the last section. First of all instead of approximating each cut $\boldsymbol{g}^{(i)}$ individually via independent sampling, we maintain a basis $\boldsymbol{B}$ that is shared by all cuts in their approximation. As a consequence instead of having a fixed approximation $\hat{\boldsymbol{g}}^{(i)}$ to each cut $\boldsymbol{g}^{(i)}$, in each iteration of the loop we have a new and better approximation $\hat{\boldsymbol{g}}^{(i)}$ by projecting it onto the new basis (which necessarily reduces the residual error). The most crucial choice in this algorithm is how to extend the basis $\boldsymbol{B}$, and we will discuss below a greedy nonlinear optimization procedure that works very well in practice.

6.5.2 Projection

The projection in Line 16 can be solved by solving the least squares problem

$$\begin{aligned}\hat{\boldsymbol{\beta}} &= \operatorname{argmin}_{\boldsymbol{\beta}} \|\boldsymbol{g}^{(t)} - \sum_{j=1}^{k} \beta_j \boldsymbol{\phi}(\boldsymbol{b}_j)\|^2 \\ &= \operatorname{argmin}_{\boldsymbol{\beta}} [\sum_{i=1}^{k}\sum_{j=1}^{k} \beta_i \beta_j K(\boldsymbol{b}_i, \boldsymbol{b}_j) - 2\sum_{j=1}^{k}\sum_{i=1}^{n} \beta_j \frac{1}{2n}(y_i - \bar{y}_i) K(\boldsymbol{x}_i, \boldsymbol{b}_j)]\end{aligned}$$

and

$$\hat{\boldsymbol{g}}^{(i)} = \sum_{j=1}^{k} \hat{\beta}_j \boldsymbol{\phi}(\boldsymbol{b}_j).$$

For convenience we denote the projection of any $\boldsymbol{g} \in \mathcal{H}$ onto the basis $\boldsymbol{B}$ as $P_{\boldsymbol{B}}\boldsymbol{g}$, where $P_{\boldsymbol{B}}$ is the projection operator. Again as in in the sampling approximation we can store the kernel products $K(\boldsymbol{x}_i, \boldsymbol{b}_j)$ in an $n \times k$ matrix $\boldsymbol{A}$. If we let $\boldsymbol{G}$ be a $k \times k$ matrix with $\boldsymbol{G}_{ij} = K(\boldsymbol{b}_i, \boldsymbol{b}_j)$, and let $\boldsymbol{Y}$ be an $n \times 1$ vector with $\boldsymbol{Y}_i = |y_i - \bar{y}_i|$,

then the coefficients $\boldsymbol{\beta}$ can be recovered by solving the linear system:

$$\boldsymbol{G}\boldsymbol{\beta} = \frac{1}{n}\boldsymbol{A}^T\boldsymbol{Y}.$$

Computing the product $\boldsymbol{A}^T\boldsymbol{Y}$ takes $O(nk)$ time while solving the linear system takes $O(k^2)$ time if we maintain a Cholesky decomposition $\boldsymbol{G} = \boldsymbol{L}_G^T\boldsymbol{L}_G$ for back substitution. The matrix $\boldsymbol{A}$ and the Cholesky factorization $\boldsymbol{L}_G$ are updated every iteration and the update costs are not counted here.

6.5.3 Basis Extension

Now it remains to specify how to extend the basis functions in Algorithm 6.3. One of the criterion of a good basis is that difference between the exact cut $\boldsymbol{g}^{(t)}$ and its approximation $\hat{\boldsymbol{g}}^{(t)}$ is "small" for all t. In this case we can conveniently measure the approximation difference using the l^2-norm. We can state our goal as finding a basis $\boldsymbol{B} = \{\boldsymbol{\phi}(\boldsymbol{b}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{b}_k)\}$ such that for all t

$$\begin{aligned} &\|\boldsymbol{g}^{(t)} - \hat{\boldsymbol{g}}^{(t)}\|^2 \leq \delta \\ \Leftrightarrow &\|\boldsymbol{g}^{(t)} - P_{\boldsymbol{B}}\boldsymbol{g}^{(t)}\|^2 \leq \delta \\ \Leftrightarrow &\min_{\boldsymbol{\beta}} \|\boldsymbol{g}^{(t)} - \sum_i \beta_i \boldsymbol{\phi}(\boldsymbol{b}_i)\|^2 \leq \delta \end{aligned} \tag{6.8}$$

for some small $\delta > 0$. We want δ to be as small as possible to keep the approximation error low.

Since we are adding one cut per iteration, it would be most natural to also grow our basis incrementally. We adopt a greedy approach as in [135]. We find the basis $\boldsymbol{b}_k$ that minimizes the residual error of the current projection $\hat{\boldsymbol{g}}^{(t)}$ onto $\boldsymbol{B} = \{\boldsymbol{\phi}(\boldsymbol{b}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{b}_k)\}$:

$$\text{argmin}_{\beta, \boldsymbol{b}_{k+1}} \|\boldsymbol{g}^{(t)} - \hat{\boldsymbol{g}}^{(t)} - \beta\boldsymbol{\phi}(\boldsymbol{b}_{k+1})\|^2. \tag{6.9}$$

This optimization problem is usually referred to as the *preimage problem* in the kernel methods literature. It is greedy in the sense that we only base our choice of $\boldsymbol{b}_{k+1}$on the single latest cut $\boldsymbol{g}^{(t)}$. But this results in a much simpler optimization problem in Equation (6.9). For differentiable kernels such as the Gaussian kernel and the polynomial kernels Equation (6.9) is a differentiable optimization problem with $d+1$ variables (for $\boldsymbol{b} \in \mathbb{R}^d$). The optimization problem is non-convex but local minima can be found by gradient based methods. In this work we use the fixed-point iteration approach in [19], which allows us to efficiently find an approximate solution to add as new basis.

To fully evaluate the benefits of choosing arbitrary vector $\boldsymbol{b} \in \mathbb{R}^d$ in our basis as opposed to the more traditional approach of picking example patterns $\boldsymbol{x}_i$ directly from the training set, we also consider solving Equation (6.9) using candidates from $\boldsymbol{b} \in \{\boldsymbol{x}_i\}_{i=1}^n$ only. We adopt the random sampling procedure by Smola and Schölkopf [119] by randomly picking 59 examples $\boldsymbol{x}_i$ and pick the one that minimizes Equation (6.9). This random sampling approach gives us 95% confidence that the basis $\boldsymbol{x}_j$ we added is within top 5% over all $\boldsymbol{x}_i$, $1 \leq i \leq n$, when evaluated over the objective in Equation (6.9) (since probability of not picking anything from the top 5% in 59 trials is $(1-0.05)^{59} < 0.05$). This random sampling procedure is also used by many other algorithms that we compare against.

6.5.4 Theoretical Analysis

Before evaluating the CPSP algorithm empirically, we first give a theoretical characterization of the quality of its solutions and the number of iterations it

takes until convergence.

The following theorem gives an upper bound on the number of iterations of Algorithm 6.3. It extends the general results [66, 67, 128] for cutting plane training of SVMs to the CPSP algorithm.

Theorem 6.3. *For parameter C, precision ϵ, training set size n, and basis set size k_{max}, Algorithm 6.3 terminates after at most $O(k_{max} + \frac{C}{\epsilon})$ iterations.*

Proof. After the first k_{max} iterations, the basis $\boldsymbol{B}$ becomes fixed, and from then on we are essentially solving the optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},\xi} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\xi \\ s.t. \quad & \forall \bar{y} \in \{-1,1\}^n : \boldsymbol{w} \cdot \left(\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)\boldsymbol{\phi}(\boldsymbol{x}_i) \right) \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i) - \xi \text{ and } \boldsymbol{w} \in \sum_{\boldsymbol{b}_i \in \boldsymbol{B}} \beta_i \boldsymbol{\phi}(\boldsymbol{b}_i) \end{aligned} \tag{6.10}$$

Let $P_{\boldsymbol{B}}$ be the orthogonal projection operator onto the subspace spanned by $\boldsymbol{B}$. Such an orthogonal projection operator always exists in a Hilbert Space. After folding the subspace constraint into the objective by replacing $\boldsymbol{w}$ with $P_{\boldsymbol{B}}\boldsymbol{w}$, the above optimization problem can be re-written as (using the self-adjointness and linearity of $P_{\boldsymbol{B}}$):

$$\begin{aligned} & \min_{\boldsymbol{w},\xi} \frac{1}{2}\|P_{\boldsymbol{B}}\boldsymbol{w}\|^2 + C\xi \\ & s.t.\ \forall \bar{y} \in \{-1,1\}^n : \boldsymbol{w} \cdot \left(\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)P_{\boldsymbol{B}}\boldsymbol{\phi}(\boldsymbol{x}_i) \right) \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y_i,\bar{y}_i) - \xi \end{aligned}$$

Finally the operator $P_{\boldsymbol{B}}$ in the objective can be dropped since if $\boldsymbol{w}$ contains any component in $\boldsymbol{B}^{\perp}$, it will only increase the objective without changing value of the LHS of the constraints. This is in the form of the general structural SVM optimization problem solved by Algorithm 3.1, with the feature space changed from being spanned by $\boldsymbol{\phi}(\boldsymbol{x}_i)$ to being spanned by $P_B\boldsymbol{\phi}(\boldsymbol{x}_i)$. The $O(\frac{C}{\epsilon})$ iteration bound from Theorem 3.1 therefore applies. □

The time complexity of each iteration was already discussed in Section 6.5.2, but can be summarized as follows. In iterations where no new basis vector is added to $\boldsymbol{B}$, the time complexity is $O(T^3 + Tk^2 + kn)$ (T being the size of the active cut set), since only the new $\boldsymbol{g}^{(t)}$ needs to be projected and the respective column be added to $\boldsymbol{H}$. In iterations where $\boldsymbol{B}$ is extended, the time complexity is $O(T^3 + k^2T + kT^2 + kTn)$ plus the time it takes to solve the preimage problem Equation (6.9). Note that typical values are $T \approx 30$, $k \in [10..1000]$, and $n > 10000$.

The following theorem describes the quality of the solution at termination, accounting for the error incurred by projecting on an imperfect $\boldsymbol{B}$. Most importantly, the theorem justifies our use of Equation (6.9) for deciding which basis vectors to add.

Theorem 6.4. *When Algorithm 6.3 terminates in m iterations with $||\boldsymbol{g}^{(i)} - \hat{\boldsymbol{g}}^{(i)}|| \leq \delta$ for all $\boldsymbol{g}^{(i)}$ and $\hat{\boldsymbol{g}}^{(i)}$, the primal objective value o of the solution found does not exceed the exact solution o^* by more than $o - o^* \leq C(\delta\sqrt{2C} + \epsilon)$.*

Proof. Let $\boldsymbol{w}^*$ be the optimal solution with value o^*. We know that the optimal $\boldsymbol{w}^*$ satisfies $\|\boldsymbol{w}^*\| \leq \sqrt{2C}$. Hence for all i,

$$|\boldsymbol{w} \cdot \boldsymbol{g}^{(i)} - \boldsymbol{w} \cdot \hat{\boldsymbol{g}}^{(i)}| \leq \|\boldsymbol{w}\|\|\boldsymbol{g}^{(i)} - \hat{\boldsymbol{g}}^{(i)}\| \leq \delta\sqrt{2C}$$

Let $P_{\boldsymbol{B}}$ be the orthogonal projection on the subspace spanned by $\boldsymbol{\phi}(\boldsymbol{b}_i)$ in the final basis $\boldsymbol{B}$. Let $\boldsymbol{v}^*$ be the optimal solution to the optimization problem (6.10)

restricted to the subspace $\boldsymbol{B}$, we have:

$$
\begin{aligned}
o &\leq \frac{1}{2}\|\boldsymbol{v}^*\|^2 + C(\xi + \epsilon) \\
&= \frac{1}{2}\|\boldsymbol{v}^*\|^2 + C \max_{1\leq i\leq m}[c^{(i)} - \boldsymbol{v}^* \cdot \hat{\boldsymbol{g}}^{(i)}] + C\epsilon \\
&\leq \frac{1}{2}\|P_{\boldsymbol{B}}\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[c^{(i)} - P_{\boldsymbol{B}}\boldsymbol{w}^* \cdot \hat{\boldsymbol{g}}^{(i)}] + C\epsilon \\
&\quad \text{[since } \boldsymbol{v}^* \text{ is the optimal solution wrt the basis } \boldsymbol{B}\text{]} \\
&= \frac{1}{2}\|P_{\boldsymbol{B}}\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[c^{(i)} - \boldsymbol{w}^* \cdot P_{\boldsymbol{B}}\hat{\boldsymbol{g}}^{(i)}] + C\epsilon \\
&= \frac{1}{2}\|P_{\boldsymbol{B}}\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[c^{(i)} - \boldsymbol{w}^* \cdot \hat{\boldsymbol{g}}^{(i)}] + C\epsilon \\
&\quad \text{[}\hat{\boldsymbol{g}}^{(i)} \text{ lies in the span of } \boldsymbol{B}\text{, so } P_{\boldsymbol{B}}\hat{\boldsymbol{g}}^{(i)} = \hat{\boldsymbol{g}}^{(i)}\text{]} \\
&\leq \frac{1}{2}\|\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[c^{(i)} - \boldsymbol{w}^* \cdot \hat{\boldsymbol{g}}^{(i)}] + C\epsilon \\
&\quad \text{[}\|P_{\boldsymbol{B}}\| \leq 1 \text{ for projections]} \\
&\leq \frac{1}{2}\|\boldsymbol{w}^*\|^2 + C \max_{1\leq i\leq m}[c^{(i)} - \boldsymbol{w}^* \cdot \boldsymbol{g}^{(i)} + \delta\sqrt{2C}] + C\epsilon \\
&\leq o^* + C(\delta\sqrt{2C} + \epsilon) \qquad \square
\end{aligned}
$$

6.5.5 Experiments

Experiment Setup

We compare the CPSP algorithm with the exact solution computed by SVM^{light}, as well as approximate solutions of the Nyström method (NYSTROM) [145], the Incomplete Cholesky Factorization (INCCHOL) [46], the Core Vector Machine (CVM) [131], the Ball Vector Machine (BVM) [130], and LASVM with margin-based active selection and finishing [14]. Both the Nyström method and the Incomplete Cholesky Factorization are implemented in SVM^{perf} as described in [67]. We use the RBF-Kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma||\boldsymbol{x} - \boldsymbol{x}'||^2)$ in all experiments.

The cache sizes of SVM^{light}, CVM, BVM, and LASVM were set to 1Gb. To evaluate the benefits of selecting general basis function $\boldsymbol{b} \in \mathbb{R}^d$, we also compare against the greedy sampling approach which selects basis functions from the training patterns $\boldsymbol{x}_i$ only. We denote the method as CPSP(tr).

We compare on the following five binary classification tasks, each split into training/validation/test set. If not mentioned otherwise, parameters (i.e. C and γ) are selected to maximize performance on the validation set for each method and k_{max} individually. Both C and γ are explored on a log-scale. The first dataset is ADULT as compiled by John Platt with 123 features and using a train/validation/test split of 20000/6281/6280. Second is the Reuters RCV1 CCAT text-classification dataset with 47236 features. We use 78127 examples from the original test set for training and split the original training set into validation and test sets of sizes 11575 and 11574 respectively. Third and fourth, we classify the digit "0" against the rest (OCR0), as well as classify the digits "01234" against the digits "56789" (OCR*) on the MNIST dataset. The MNIST datasets have 780 features and we use a training/validation/test split of 50000/5000/5000. Finally, we use the IJCNN (task 1) dataset as pre-processed by Chih-Jen Lin. It has 22 features and we use a training/validation/test split of 113533/14169/14169.

Accuracy with Fixed Number of Basis Functions

Table 6.2 shows the performance of different algorithms with the number of basis functions fixed at 1000. The results from SVM^{light} are for reference. We can see that except for the ADULT dataset, the Gaussian (RBF) kernel SVM has higher test set accuracy than the linear SVM. The table also shows the number

Table 6.2: Prediction accuracy with $k_{max} = 1000$ basis vectors (except SVM^{light}, where the number of SVs is shown in the third line) using the RBF kernel (except linear).

	ADULT	CCAT	OCR0	OCR*	IJCNN
SVM^{light} (linear)	84.4	94.2	99.4	87.6	92.2
SVM^{light} (RBF)	84.4	95.1	99.8	98.6	99.4
#SV	7125	28748	2786	19309	9243
CPSP	**84.5**	**95.0**	**99.8**	**98.5**	**99.3**
CPSP(tr)	84.1	93.5	**99.8**	97.9	99.2
NYSTROM	84.3	92.5	99.7	97.0	99.1
INCCHOL	84.0	92.1	99.7	97.0	98.9
CVM	78.4	88.1	**99.8**	96.9	98.2
BVM	77.1	56.1	**99.8**	89.1	97.7
LASVM	83.8	91.7	**99.8**	97.2	97.5

of support vectors (#SV) for the Gaussian kernel. We can see that for datasets like ADULT, CCAT, OCR*, well over one third of the training examples become support vectors. IJCNN also have about 10% of the training examples become support vectors while OCR0 is an easier task with the examples more separable, and hence there are fewer support vectors in that case. We can see that the size of the support vector set is large and restricting the basis function set size to 1000 can significantly improve the classification time.

Let us examine the accuracies of each of the algorithm when restricted to 1000 basis functions. We can see that CPSP has the highest accuracy across all 5 datasets, and is always no more than 0.1% worse than the exact solution by SVM^{light}. We also observe that by comparing CPSP against CPSP(tr), us-

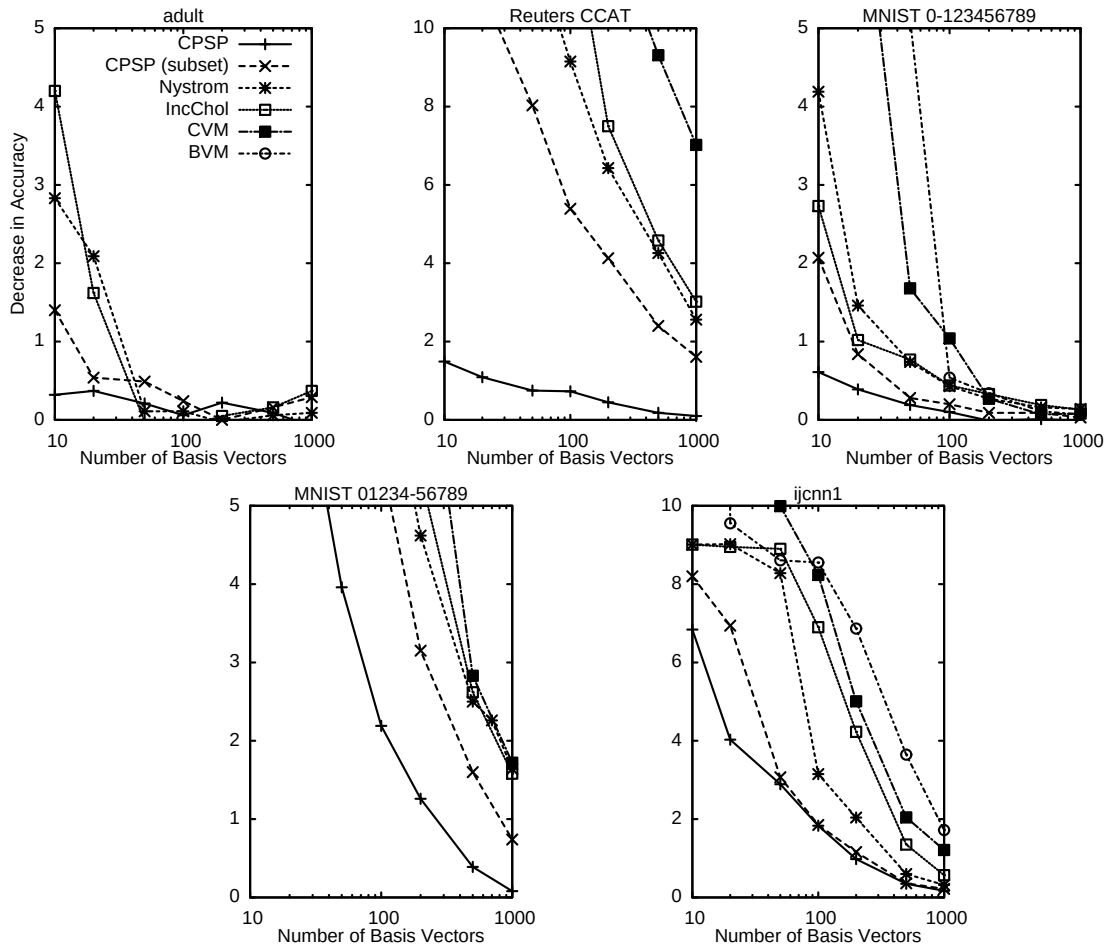


Figure 6.7: Decrease in accuracy w.r.t. exact SVM for different basis-set sizes k_{max}.

ing general basis vectors $\boldsymbol{b} \in \mathbb{R}^d$ improves the test accuracy when given the same budget for basis functions, especially on the CCAT and OCR* datasets. CPSP has better test accuracies than competitors such as NYSTROM, INCCHOL, LASVM by 1 to 2% depending on the dataset, while CVM and BVM have stability issues with some datasets and give substantially lower accuracy numbers on ADULT and CCAT.

Accuracy-Sparsity Tradeoff

Our next set of experiments look at how the test set accuracy changes when we vary the size of the basis set. In Figure 6.7 we plot the decrease in test set accuracy against the basis set size, from 10 to 1000. The decrease in test set accuracy is measured against the exact solution by SVM^{light}. These plots allow us to observe the tradeoff between test accuracy and sparsity levels. We can see that for all the datasets except ADULT when the number of basis functions is large (in which case all algorithms are close), the curves for CPSP and CPSP(tr) lies completely below their competitors (lower is better). This shows across a range of basis set size, CPSP and CPSP(tr) has higher accuracies than their competitors. The difference is particularly large when we look at the range from 10 to 100 basis functions. Also when we compare CPSP against CPSP(tr), we observe that the curve CPSP dominates the curve for CPSP(tr) for the accuracy-sparsity tradeoff. This validates our hypothesis that using basis vectors $\boldsymbol{b} \in \mathbb{R}^d$ outside the training set improves sparsity. This is particularly evident in the CCAT text categorization dataset, where we can observe the decrease in accuracy being more than halfed by using general basis vectors.

Training and Test Efficiency

Table 6.3 shows the number of basis functions required to reach within 0.5% accuracy of the exact solution for different algorithms and Table 6.4 shows their corresponding training time. Basis set sizes are fixed at $\{10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000\}$ for each method, and the smallest basis set size that reached the required accuracy is shown. Like the previous results CPSP requires the fewest basis functions. When compared

Table 6.3: Number of SVs to reach an accuracy that is not more than 0.5% below the accuracy of the exact solution of SVM^{light}(see Table 6.2). The RBF kernel is used for all methods. '>' indicates that the largest tractable solution did not achieve the target accuracy.

	Number of SV				
	ADULT	CCAT	OCR0	OCR*	IJCNN
SVM^{light}	7125	28748	2786	19309	9243
CPSP	**10**	**200**	**20**	**500**	**500**
CPSP(tr)	50	5000	50	2000	**500**
NYSTROM	50	>5000	100	5000	1000
INCCHOL	50	>2000	100	>2000	2000
CVM	5000	20000	200	5000	2000
BVM	5000	20000	200	5000	5000
LASVM	2000	10000	100	2000	5000

against the exact solution we have about 18 to 712 times fewer support vectors, translating directly into a 18 to 712 times speedup in classification during test time. Even when compared against other fast kernel SVM algorithms we usually have a 5 to 10 times improvement in sparsity/test time speedup. Table 6.4 shows the training time of the different algorithms. Unlike the number of basis functions there is no clear winner in terms of training time across all 5 datasets. With the exception of IJCNN, CPSP is amongst the fastest, being the fastest for ADULT and CCAT and not far behind the fastest method for OCR* and OCR0. The possible overhead during training for CPSP is the solution of the preimage problem in Equation (6.9) and re-projecting the cuts onto the new basis, but overall CPSP has very fast training time that puts it right among the fastest training algorithms for nonlinear kernel SVMs.

Table 6.4: Training time to reach an accuracy that is not more than 0.5% below the accuracy of the exact solution of SVM^{light}(see Table 6.2). The RBF kernel is used for all methods. '>' indicates that the largest tractable solution did not achieve the target accuracy.

	CPU-Seconds				
	ADULT	CCAT	OCR0	OCR*	IJCNN
SVM^{light}	56	9272	400	4629	1175
CPSP	**6**	**225**	11	465	2728
CPSP(tr)	30	88873	57	8967	2178
NYSTROM	10	>2281	37	2270	1572
INCCHOL	14	>21673	66	>12330	59454
CVM	43	23730	**2**	497	**29**
BVM	67	11004	**2**	538	229
LASVM	51	3433	5	**295**	705

6.6 Applications in General Structural SVMs

The above approximation methods using sampling and preimage optimization can be easily extended to general structural SVMs. For example, consider again the MRF labeling example in Equation (6.5). The cutting planes in this case will be linear combinations of the joint feature vectors in the training sample, and is of the following form:

$$\begin{aligned}\boldsymbol{g} &= \frac{1}{n}\sum_{i=1}^{n}[\boldsymbol{\Phi}(x_i, y_i) - \boldsymbol{\Phi}(x_i, \bar{y}_i)] \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[\left(\sum_{(u,v)\in E(x_i)}\boldsymbol{\phi}_{uv}(x_{i,u}, x_{i,v}, y_{i,u}, y_{i,v}) + \sum_{u\in V(x_i)}\boldsymbol{\phi}_u(x_{i,u}, y_{i,u})\right)\right. \\ &\quad \left. - \left(\sum_{(u,v)\in E(x_i)}\boldsymbol{\phi}_{uv}(x_{i,u}, x_{i,v}, \bar{y}_{i,u}, \bar{y}_{i,v}) + \sum_{u\in V(x_i)}\boldsymbol{\phi}_u(x_{i,u}, \bar{y}_{i,u})\right)\right],\end{aligned}$$

where $x_{i,u}$, $y_{i,u}$ are the corresponding local features and label at the node u in the MRF defined by x_i. Suppose the edge features $\boldsymbol{\phi}_{uv}$ are linear while the node features $\boldsymbol{\phi}_u$ are kernelized (with polynomial kernels, RBF kernels, etc). We can very easily collect the nonlinear terms in the cutting plane

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{u\in V(x_i)}[\boldsymbol{\phi}_u(x_{i,u},y_{i,u})-\boldsymbol{\phi}_u(x_{i,u},\bar{y}_{i,u})], \tag{6.11}$$

and then apply either the sampling procedure or preimage optimization technique in this chapter before adding the cut to the working set. For the linear part of the feature vector $\boldsymbol{\phi}_{uv}$ we can represent it exactly and there is no need for approximation. Note that in the sampling approximation algorithm we made no assumption about the training examples x_i coming from an i.i.d. distribution, and the randomness in the Hoeffding bound comes from the uniform sampling procedure itself. Therefore the approximation guarantees still hold when we are sampling at the node level for $\boldsymbol{\phi}_u(x_{i,u},y_{i,u})$, and there is no need to worry about the dependence relations between nodes in the graph of the MRF instance x_i. Similarly in the projection and nonlinear preimage optimization in the CPSP algorithm we can just treat the nonlinear part of the cutting plane (Equation (6.11)) as we did in the case for binary classification.

6.7 Conclusions

We have proposed two approximation methods based on the cutting plane algorithm for training nonlinear kernel SVMs. The methods are equally suitable to handle the use of nonlinear kernels in structural SVMs. Experimental evaluations show that the methods have improved training speed, and the preimage method also produced sparser solutions when compared to other approxima-

tion algorithms for training kernel SVMs.

6.8 Bibliographical Notes

These works on approximating cutting planes for training nonlinear kernel SVMs were published at [150] (sampling) and at [68] (preimage optimization).

There have been a lot of different works on speeding up the training of nonlinear kernel SVMs and other kernel methods. The Nyström method [145], Incomplete Cholesky Factorization [46] and Sparse Greedy Matrix Approximation all try to approximate the kernel matrix with a low-rank subspace and then solve the corresponding kernel SVM in this subspace. However these approximation methods focus on the reconstruction error of the matrix but do not take into account the label information, with the exception of the work by Bach and Jordan [9]. Unlike CPSP the approximation step is not integrated into the optimization step of the learning process , and these methods are limited to choosing basis vectors from the training set.

Core Vector Machine [131] and Ball Vector Machine [130] are approximation algorithms based on ideas from computational geometry [10], and the problem of solving an l^2-penalized kernel SVM is reduced to approximating the center and radius of a ball enclosing the data points in the RKHS. LASVM [14] is an online algorithm that adds and remove basis functions greedily to train a kernel SVM in very few passes over the data. Basis pursuit style algorithms such as [74] and [135] try to incrementally add or remove basis function from a basis set and then re-solve the whole optimization problem to produce a sparse basis set with good objective values. However unlike CPSP all these methods focus on

selecting basis vectors from the training set alone.

Methods that allows the use of general basis vector includes the Reduced Set Method [19] and the nonlinear optimization approach by Wu et al. [146]. The Reduced Set Method is a postprocessing method that sparsifies the kernel representation of the weight vector $\boldsymbol{w}$ after the kernel SVM is solved, and does not address the problem of long training time for kernel SVMs. The approach by Wu et al. casts the kernel SVM learning problem as a large joint non-convex optimization problem over the weight vector and the basis vector representation. Their method is not as scalable as CPSP and most of their evaluations were on small training sets.

As for sampling based stochastic approximations, most current works focus on linear feature space. For example, [137] consider the use of stochastic gradient descent in the training of CRF and the PEGASOS algorithm [114] uses stochastic subgradient to train linear SVMs.

CHAPTER 7

CONCLUSIONS AND FUTURE DIRECTIONS

In this chapter we are going to summarize what we have presented in this thesis, and also sketch a few future directions for further research in the area of structured output learning, especially related to large-margin methods such as structural SVMs.

7.1 Conclusions

In this thesis we have presented structural support vector machines, a discriminative large-margin method for structured output learnng. In particular, we have introduced extensions to structural SVMs that allow them to handle latent variables and for nonlinear kernels to be incorporated efficiently. Below is a chapter-by-chapter summary of what we have presented in this thesis.

Chapter 2 is a general introduction to structured output learning and structural SVMs. Starting from binary classification SVMs, we described how we can generalize from binary SVMs to multi-class SVMs and then to structural SVMs through refining the constraints in a quadratic program. Using part-of-speech tagging as a running example, we illustrated how we can estimate the parameters in a traditional hidden Markov model generatively, and also how the paramters can be estimated differently in a discriminative manner using structural SVMs. In particular we emphasized the ease with which we can introduce new features based on the input x to improve prediction performance, which is one of the major benefits of discriminative modeling.

Chapter 3 is concerned with the training of structural SVMs, which involves

solving a nonsmooth convex optimization problem. In particular we presented the cutting plane algorithm proposed in [67] for training structural SVMs. We also presented the $O(1/\epsilon)$ iteration bound argument for the cutting plane algorithm in [67]. One of the main computation in the algorithm is the generation of cutting planes, and we illustrated how it can be done with the loss-augmented inference in structural SVMs. We also discussed what design decisions one has to make when adapting structural SVMs to different applications.

Chapter 4 illustrates the application of structural SVM to a protein alignment problem. Structural SVM is a very suitable method for the problem because the discriminative training approach allows us to fully exploit the sequence and structural features in the protein database to learn alignments. We illustrated concretely how to design the joint feature maps and loss functions in structural SVM, and the associated inference algorithms for cutting plane algorithm training. Experimental results showed that we can build highly complex alignment models incorporating many features that have improved alignment accuracies over generative models.

In Chapter 5 we discussed the problem of latent variables in structured output learning. We proposed a simple yet general extension to structural SVM that allows us to handle latent variables in large margin structured output learning, which we called Latent Structural SVM. We made use of the Convex-Concave Procedure [156] to solve the corresponding non-convex optimization problem. The latent structural SVM formulation is as easy to use as standard structural SVM, and we demonstrated how to design the joint feature maps, loss functions, and associated inference procedures in three different applications including motif finding, noun phrase coreference resolution, and optimizing for

precision@k in information retrieval. Experimental results showed improved prediction accuracies in the structured output learning tasks when they are re-formulated with latent variables, and in some cases the re-formulation also brings about simpler inference procedures with improved training speed.

In the last chapter, Chapter 6, we looked at the use of nonlinear kernels in structural SVMs. Nonlinear kernels increase the expressiveness of structural SVMs, but also greatly increase its training costs to $O(n^2)$, where n is the number of examples. We presented two approximation algorithms based on the cutting plane algorithm in Chapter 3, one using sampling and one using nonlinear preimage optimzation. We provided iteration bounds and approximation error bounds on the objectives. When evaluated over binary classification problems, both algorithms have improved training speed over conventional dual decomposition based algorithms such as SVM^{light}. The nonlinear preimage optimization also have improved sparsity-accuracy tradeoff when compared against approximation algorithms designed for speeding up the training of binary kernel SVMs in the literature. We also illustrated how these cutting plane approximation ideas can be easily applied to general structured output learning settings where the individual feature functions can be 'kernelized' to become nonlinear.

In the above we summarized some of the contributions we made to discriminative large-margin structured output learning in this thesis, but looking forward there are still a lot of important challenges. Below we sketch some interesting directions for future work.

7.2 Future Directions

7.2.1 Approximate Inference for Training

As mentioned in Chapter 3, the most expensive step in training structural SVMs is solving the argmax problem in Equation (2.10) for cutting plane generation. For many structured output prediction problem such as labeling Markov Random Field with cycles [48], decoding in machine translation [78], or maximizing diversity/topic coverage in document retrieval [155], the related argmax problems are NP-complete. Researchers have looked into the use of various approximation algorithms in training [80, 48, 89], such as loopy belief propagation, linear programming relaxation, or any problem-specific greedy approximation or heuristic search. These ideas make structured output prediction learning possible for these tasks with intractable inference problems. However, less well-explored is the use of these approximation ideas to speed up training in structured output learning problems with tractable/polynomial inference problems. For example, although problems like POS-tagging, parsing, protein sequence alignments have dynamic programming based algorithms to exactly solve the argmax inference problems, training is still very slow on large datasets because these inference algorithms scale with $O(L^2)$ or $O(L^3)$, where L is the size of each example (e.g., sequence length). If approximate inference can be applied to speed up the training of models with NP-hard inference problems, can we apply similiar ideas to speed up the training of these problems with polynomial but still expensive inference operations?

Consider again our very first motivating example of structured output prediction with POS-tagging. We know that in POS-tagging it is important to con-

sider the context of the word to determine its part-of-speech, however for many words in the dictionary they have only one possible part-of-speech and the decision can be made without looking at the context at all. As far as the learning algorithm is concerned there is no need for it to infer the tags in the whole sentence to learn a particular word has only one possible part-of-speech. Indeed in a lot of structured output prediction problems many decisions can be made by considering only a local part of the structure. Can we apply a coarse-grained, approximate inference procedure at the beginning of the training, and then slowly switch to a fine-grained, exact inference procedure when we close to the optimal solution of the structural SVM training problem? A lot of inference algorithms such as heuristic search, primal-dual algorithms for combinatorial optimization gives us partial, incomplete output structure during the inference process. Can we use these partial, incomplete outputs to provide feedback for the optimization algorithms to tune the weights? For example, it would be highly beneficial to use such partial outputs to construct cutting planes during the initial iterations of the cutting plane algorithm to reduce runtime, since these initial cutting planes are rarely active at the optimum.

7.2.2 Parallelization

Another direction to reduce the training time of these discriminative structured output prediction models is through the use of parallelization. In recent years the lack of improvements in the clock speed of processors and the increasing availability of highly parallel multi-core processors and graphical processing unit make parallelization an important topic in speeding up practical algorithms. Let us consider the problem of training structured output prediction

models such as structural SVMs and conditional random fields, where the risk is sum of individual loss on each example (x_i, y_i). It is quite easy to parallelize the problem by distributing the gradient or cutting plane computation on different processors or different machines, by making each processor compute the gradient on a subset of the examples and then add up the results. However there are still multiple issues on parallelization that are worth investigating. First, as [88] pointed out gradient based algorithms can incur large communication costs between machines during parallelization, both in terms of the number synchronizations required and also the volume of data exchanged. The problem becomes even more severe if we consider nonlinear models with kernels. Second, although it is relatively straightforward to parallelize gradient computation by distributing the individual training instances (embarassingly parallel), breaking down the gradient computation on a single instance (x_i, y_i) is much more difficult. For example, inference in large scale graphical models or heuristic-based search are much more difficult to parallelize. There are already some works looking at these directions [51]. Building a better theoretical and empirical understanding of how efficiently we can parallelize these computations will be important for structured output learning.

7.2.3 Reducing Dependence on Labeled Data

Apart from training efficiency, another major issue in discriminative structured output learning is the lack of labeled training data. Although discriminative modeling gives us the power to incorporate a large number of features to improve prediction accuracy, at the same time we need to obtain sufficient labeled data to train these models. Unlike binary or multi-class classification the la-

beling effort for structured output prediction is much more expensive. For example, manually labeling the pixels for image segmentation or labeling all the objects inside a picture is a very tedious task. Some tasks such as machine translation even requires specially trained individuals who are fluent in both languages to provide us with the labels. In semi-supervised learning for classification, we try to incorporate unlabeled examples x_j without the corresponding labels y_j to improve the prediction accuracy of models trained with labeled data alone (see [25] for a review of literature in the area). It is quite natural to ask if we could reduce the labeling effort in training these discriminative structured output learning models as well. However, success in this area has been much more limited than the binary classification case. Direct generalizations of methods from semi-supervised binary classicification methods only help when there are very few labeled data [4, 158]. More promising approaches include defining multiple related learning tasks to make use of the unlabeled data [7], and exploiting generative models to define new features in discriminative modeling [124]. However, most of these involve careful design specific to the computer vision or natural language processing application, and it will be interesting if we could abstract away some general principles of making use of unlabeled data in structured output prediction from these application experiences.

Another interesting direction of making use of unlabeled data in structured output learning is the role of unlabeled data in the output space $\mathcal{Y}$. Unlike in the case of classification where the output space consists of class labels with little structures, the output space $\mathcal{Y}$ in structured output learning has rich structure that we can estimate using data in $\mathcal{Y}$ only. Intuitively in discriminative modeling where we try to model the conditional distribution $P(y \mid x)$, knowledge about $P(y)$ should help us improve the estimation of $P(y \mid x)$. A prime example of

this is statistical machine translation, in which the output space $\mathcal{Y}$ is a set of sentences in the target language that is high dimensional and have a lot of rich structures in it. Knowing how to exploit these structures with unlabeled data in the output space $\mathcal{Y}$ should improve the prediction accuracies of structured output learning models, and this could apply to other tasks in computer vision and natural language processing as well.

BIBLIOGRAPHY

[1] R. Adamczak, A. Porollo, and J. Meller. Accurate prediction of solvent accessibility using neural networks-based regression. *Proteins: Structure, Function, and Bioinformatics*, 56:753–767, 2004.

[2] M.A. Aizerman, E.M. Braverman, and L. Rozonoèr. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25(6):821–837, 1964.

[3] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389, 1997.

[4] Y. Altun, D. McAllester, and M. Belkin. Maximum margin semi-supervised learning for structured variables. *Advances in Neural Information Processing Systems*, 18:33, 2006.

[5] Y. Altun and A. Smola. Unifying divergence minimization and statistical inference via convex duality. In *Proceedings of the 19th Conference on Learning Theory (COLT)*, pages 139–153. Springer, 2006.

[6] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden Markov support vector machines. In *Proceedings of the 20th International Conference on Machine Learning*, volume 20, page 3, 2003.

[7] R.K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *The Journal of Machine Learning Research*, 6:1817–1853, 2005.

[8] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

[9] F.R. Bach and M.I. Jordan. Predictive low-rank decomposition for kernel methods. In *Proceedings of the 22nd International Conference on Machine Learning*, page 40. ACM, 2005.

[10] M. Bādoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, page 257. ACM, 2002.

[11] T.L. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the 2nd International Conference on Intelligent Systems Molecular Biology*, volume 2, pages 28–36, 1994.

[12] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[13] P. Blunsom and T. Cohn. Discriminative word alignment with conditional random fields. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, page 72. Association for Computational Linguistics, 2006.

[14] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *The Journal of Machine Learning Research*, 6:1619, 2005.

[15] S.P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[16] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1124–1137, 2004.

[17] P.F. Brown, V.J.D. Pietra, S.A.D. Pietra, and R.L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

[18] R.C. Bunescu and R.J. Mooney. Multiple instance learning for sparse positive bags. In *Proceedings of the 24th International Conference on Machine Learning*, page 112. ACM, 2007.

[19] C.J.C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. *Advances in Neural Information Processing Systems*, pages 375–381, 1997.

[20] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the 13th ACM international Conference on Information and Knowledge Management*, pages 78–87. ACM, 2004.

[21] Z. Cao, T. Qin, T.Y. Liu, M.F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136. ACM, 2007.

[22] C.S. Chan and B.K. Tye. Autonomously replicating sequences in Saccharomyces cerevisiae. *Proceedings of the National Academy of Sciences*, 77(11):6329, 1980.

[23] O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

[24] O. Chapelle, C. Do, Q. Le, A. Smola, and C. Teo. Tighter bounds for structured estimation. In *Advances in Neural Information Processing Systems*, pages 281–288, 2008.

[25] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-supervised learning*. MIT Press, 2006.

[26] S.S. Chen. *Basis pursuit*. PhD thesis, Department of Statistics, Stanford University, 1995.

[27] C. Cherry and C. Quirk. Discriminative, syntactic language modeling through latent SVMs. *Proceeding of Association for Machine Translation in the America (AMTA-2008)*, 2008.

[28] P.M. Cheung and J.T. Kwok. A regularization framework for multiple-instance learning. In *Proceedings of the 23rd International Conference on Machine Learning*, page 200. ACM, 2006.

[29] M. Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics, 1997.

[30] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing-Volume 10*, page 8. Association for Computational Linguistics, 2002.

[31] M. Collins and N. Duffy. Convolution kernels for natural language. *Advances in Neural Information Processing Systems*, 1:625–632, 2002.

[32] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. In *Advances in Neural Information Processing Systems*, 2002.

[33] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proceedings of the 23rd International Conference on Machine Learning*, page 208. ACM, 2006.

[34] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[35] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002.

[36] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991, 2003.

[37] J.N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480, 1972.

[38] H. Daumé, J. Langford, and D. Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[39] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5(Suppl 3):345–352, 1978.

[40] A.P. Dempster, N.M. Laird, D.B. Rubin, et al. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[41] C. Do, S. Gross, and S. Batzoglou. CONTRAlign: discriminative training for protein sequence alignment. In *Research in Computational Molecular Biology*, pages 160–174. Springer, 2006.

[42] T.M.T. Do and T. Artières. Large margin training for hidden Markov models with partially observed states. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 265–272. ACM, 2009.

[43] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge University Press, 2002.

[44] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[45] P. Felzenszwalb, D. McAllester, and D. Ramanan. A Discriminatively Trained, Multiscale, Deformable Part Model. In *Proceedings of the IEEE CVPR*, pages 1–8, 2008.

[46] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *The Journal of Machine Learning Research*, 2:243–264, 2002.

[47] T. Finley and T. Joachims. Supervised clustering with support vector machines. In *Proceedings of the 22nd International Conference on Machine Learning*, page 224. ACM, 2005.

[48] T. Finley and T. Joachims. Training structural SVMs when exact inference is intractable. In *Proceedings of the 25th International Conference on Machine Learning*, pages 304–311. ACM, 2008.

[49] V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25th International Conference on Machine Learning*, pages 320–327. ACM, 2008.

[50] B. Gassend, CW O'Donnell, W. Thies, A. Lee, M. Van Dijk, and S. Devadas. Secondary Structure Prediction of All-Helical Proteins Using Hidden Markov Support Vector Machines. *Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory*, 2005.

[51] J. Gonzalez, Y. Low, and C. Guestrin. Residual splash for optimally parallelizing belief propagation. *In Artificial Intelligence and Statistics (AISTATS)*, pages 1–8, 2009.

[52] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society, 2005.

[53] R. Grishman and B. Sundheim. Message understanding conference-6: A brief history. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, page 471. Association for Computational Linguistics, 1996.

[54] A. Gunawardana, M. Mahajan, A. Acero, and J.C. Platt. Hidden conditional random fields for phone classification. In *Ninth European Conference on Speech Communication and Technology*. ISCA, 2005.

[55] D. Gusfield and P. Stelling. Parametric and inverse-parametric sequence alignment with XPARAL. *Methods in Enzymology*, 266:481–494, 1996.

[56] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1):389–422, 2002.

[57] D. Haussler. Convolution kernels on discrete structures. Technical report, University of California at Santa Cruz, 1999.

[58] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89(22):10915, 1992.

[59] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, chapter 7, pages 115–132. MIT Press, 2000.

[60] J.B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms: Fundamentals*. Springer, 1993.

[61] R. Horst and N.V. Thoai. DC programming: overview. *Journal of Optimization Theory and Applications*, 103(1):1–43, 1999.

[62] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98*, pages 137–142, 1998.

[63] T. Joachims. Making large scale support vector machine learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

[64] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, 2002.

[65] T. Joachims. A support vector method for multivariate performance mea-

sures. In *Proceedings of the 22nd International Conference on Machine Learning*, page 384. ACM, 2005.

[66] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 226. ACM, 2006.

[67] T. Joachims, T. Finley, and C.N.J. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59, 2009.

[68] T. Joachims and C.-N. Yu. Sparse kernel SVMs via cutting-plane training. *Machine Learning (ECML PKDD 2009 Special Issue)*, 76(2-3):179–193, 2009.

[69] D.T. Jones. GenTHREADER: an efficient and reliable protein fold recognition method for genomic sequences. *Journal of Molecular Biology*, 287(4):797–815, 1999.

[70] M.I. Jordan. *Learning in graphical models*. Kluwer Academic Publishers, 1998.

[71] W. Kabsch and C. Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983.

[72] T. Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages, 1965.

[73] J. Kececioglu and E. Kim. Simple and fast inverse alignment. In *Research in Computational Molecular Biology*, pages 441–455. Springer, 2006.

[74] S.S. Keerthi, O. Chapelle, and D. DeCoste. Building support vector machines with reduced classifier complexity. *The Journal of Machine Learning Research*, 7:1515, 2006.

[75] J.E. Kelley Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[76] G. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions* 1. *Journal of Mathematical Analysis and Applications*, 33(1):82–95, 1971.

[77] K.C. Kiwiel. *Methods of descent for nondifferentiable optimization*. Springer Berlin, 1985.

[78] K. Knight. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615, 1999.

[79] T. Koo, A. Globerson, X. Carreras, and M. Collins. Structured prediction models via the matrix-tree theorem. In *Proc. EMNLP*, 2007.

[80] A. Kulesza and F. Pereira. Structured learning with approximate inference. *Advances in Neural Information Processing Systems*, 20:785–792, 2008.

[81] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, 2001.

[82] C.E. Lawrence, S.F. Altschul, M.S. Boguski, J.S. Liu, A.F. Neuwald, and J.C. Wootton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, 1993.

[83] C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, volume 7, pages 566–575, 2002.

[84] Y. Li and D.P. Huttenlocher. Learning for stereo vision using the structured support vector machine. In *Proceedings of the IEEE CVPR*, 2008.

[85] D.C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[86] T.Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *SIGIR Workshop on Learning to Rank for Information Retrieval*, 2007.

[87] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *The Journal of Machine Learning Research*, 2:444, 2002.

[88] G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. *Advances in Neural Information Processing Systems*, 2009.

[89] A.F.T. Martins, N.A. Smith, and E.P. Xing. Polyhedral outer approximations with application to natural language parsing. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 713–720. ACM, 2009.

[90] A. McCallum, K. Bellare, and F. Pereira. A conditional random field for discriminatively-trained finite-state string edit distance. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[91] R. McDonald, K. Crammer, and F. Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, page 98. Association for Computational Linguistics, 2005.

[92] J. Meller and R. Elber. Linear programming optimization and a double statistical filter for protein threading protocols. *Proteins: Structure, Function, and Bioinformatics*, 45(3):241–261, 2001.

[93] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A*, 209:415–446, 1909.

[94] M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2004.

[95] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[96] M.E.J. Newman, D.J. Watts, and S.H. Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 99(Suppl 1):2566, 2002.

[97] P. Ng and U. Keich. GIMSAN: a Gibbs motif finder with significance analysis. *Bioinformatics*, 24(19):2256, 2008.

[98] V. Ng and C. Cardie. Improving machine learning approaches to coreference resolution. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics, 2002.

[99] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for

support vector machines. In *Neural Networks for Signal Processing VII Proceedings of the 1997 IEEE Workshop*, pages 276–285, 1997.

[100] L. Pachter and B. Sturmfelds. Parametric inference for biological sequence analysis. In *Proceedings of the National Academy of Sciences*, volume 101, pages 16138–16143, 2004.

[101] S. Petrov and D Klein. Discriminative log-linear grammars with latent variables. In *Advances in Neural Information Processing Systems*, page 1153, 2007.

[102] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 12. MIT-Press, 1999.

[103] M. Pontil and A. Verri. Support vector machines for 3D object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):637–646, 1998.

[104] J. Qiu and R. Elber. SSALN: an alignment algorithm using structure-dependent substitution matrices and gap penalties learned from structurally aligned protein pairs. *Proteins: Structure, Function, and Bioinformatics*, 62(4):881–891, 2006.

[105] L. R. Rabiner. A tutorial on hidden Markov models and selected applications inspeech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[106] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:141, 2004.

[107] R. Rosenfeld. Two decades of statistical language modeling: where do we go fromhere? *Proceedings of the IEEE*, 88(8):1270–1278, 2000.

[108] M. Saito and M. Matsumoto. SIMD-oriented fast mersenne twister: a 128-bit pseudorandom number generator. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, 2006.

[109] S. Sarawagi and R. Gupta. Accurate max-margin training for structured output spaces. In *Proceedings of the 25th International Conference on Machine Learning*, pages 888–895. ACM, 2008.

[110] B. Schölkopf, R. Herbrich, and A. Smola. A generalized representer theorem. In *Computational Learning Theory*, pages 416–426. Springer, 2001.

[111] B. Schölkopf and A.J. Smola. *Learning with kernels*. MIT Press, 2002.

[112] B. Schölkopf, K. Tsuda, and J.P. Vert. *Kernel methods in computational biology*. MIT Press, 2004.

[113] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141. Association for Computational Linguistics, 2003.

[114] S. Shalev-Shwartz, Y. Singer, and N. Srebro. PEGASOS: Primal estimated sub-gradient solver for SVM. In *Proceedings of the 24th International Conference on Machine Learning*, page 814. ACM, 2007.

[115] C.E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):55, 2001.

[116] J. Shi, T.L. Blundell, and K. Mizuguchi. FUGUE: sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *Journal of Molecular Biology*, 310(1):243–257, 2001.

[117] I.N. Shindyalov and P.E. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Engineering Design and Selection*, 11(9):739, 1998.

[118] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[119] A.J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the 7th International Conference on Machine Learning*, page 918. ACM, 2000.

[120] A.J. Smola, S.V.N. Vishwanathan, and T. Hofmann. Kernel methods for missing variables. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS)*, page 325, 2005.

[121] B.K. Sriperumbudur and G.R.G. Lanckriet. On the convergence of the

concave-convex procedure. *Advances in Neural Information Processing Systems*, 2009.

[122] I. Steinwart. Sparseness of support vector machines. *The Journal of Machine Learning Research*, 4:1071–1105, 2003.

[123] Fangting Sun, D. Fernandez-Baca, and Wei Yu. Inverse parametric sequence alignment. In *International Computing and Combinatorics Conference (COCOON)*, 2002.

[124] J. Suzuki, H. Isozaki, X. Carreras, and M. Collins. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 551–560. Association for Computational Linguistics, 2009.

[125] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *Advances in Neural Information Processing Systems*, 2003.

[126] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *Proc. EMNLP*, pages 1–8, 2004.

[127] B. Taskar, S. Lacoste-Julien, and D. Klein. A discriminative matching approach to word alignment. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 73–80. Association for Computational Linguistics, 2005.

[128] C.H. Teo, A. Smola, S.V.N. Vishwanathan, and Q.V. Le. A scalable modular convex solver for regularized risk minimization. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 736. ACM, 2007.

[129] J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673, 1994.

[130] I.W. Tsang, A. Kocsor, and J.T. Kwok. Simpler core vector machines with enclosing balls. In *Proceedings of the 24th International Conference on Machine Learning*, page 918. ACM, 2007.

[131] I.W. Tsang, J.T. Kwok, and P.M. Cheung. Core vector machines: Fast

SVM training on very large data sets. *Journal of Machine Learning Research*, 6(1):363, 2006.

[132] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning*, page 104. ACM, 2004.

[133] V.N. Vapnik. *The nature of statistical learning theory*. Springer Verlag, 2000.

[134] M. Vilain, J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman. A model-theoretic coreference scoring scheme. In *Proceedings of the 6th conference on Message understanding*, pages 45–52. Association for Computational Linguistics, 1995.

[135] P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48(1):165–187, 2002.

[136] S. V. N. Vishwanathan, A. J. Smola, and M. N. Murty. SimpleSVM. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

[137] S.V.N. Vishwanathan, N.N. Schraudolph, M.W. Schmidt, and K.P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of the 23rd International Conference on Machine Learning*, page 976. ACM, 2006.

[138] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[139] M. Wagner, J. Meller, and R. Elber. Large-scale linear programming techniques for the design of protein folding potentials. *Mathematical Programming*, 101(2):301–318, 2004.

[140] S.B. Wang, A. Quattoni, L.P. Morency, D. Demirdjian, and T. Darrell. Hidden conditional random fields for gesture recognition. In *Proceedings of the IEEE CVPR*, page 1521, 2006.

[141] Yang Wang and Greg Mori. Max-margin hidden conditional random fields for human action recognition. Technical Report TR 2008-21, School of Computing Science, Simon Fraser University, 12 2008.

[142] C. Watkins. Dynamic alignment kernels. *Advances in Neural Information Processing Systems*, pages 39–50, 1999.

[143] A. Weller, D. Ellis, and T. Jebara. Structured prediction models for chord transcription of music audio. In *2009 International Conference on Machine Learning and Applications*, pages 590–595. IEEE, 2009.

[144] J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik. Kernel dependency estimation. *Advances in Neural Information Processing Systems*, pages 897–904, 2003.

[145] C.K.I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. *Advances in Neural Information Processing Systems*, pages 682–688, 2001.

[146] M. Wu, B. Schölkopf, and G. Bakır. A direct method for building sparse kernel learning algorithms. *The Journal of Machine Learning Research*, 7:624, 2006.

[147] J. Xu, M. Li, D. Kim, and Y. Xu. RAPTOR: optimal protein threading by linear programming. *International Journal of Bioinformatics and Computational Biology*, 1(1):95–118, 2003.

[148] M.H. Yang. Kernel eigenfaces vs. kernel Fisherfaces: Face recognition using kernel methods. In *Fifth IEEE International Conference on Automatic Face and Gesture Recognition*, page 0215, 2002.

[149] D.H. Younger. Recognition and parsing of context-free languages in time n3*. *Information and control*, 10(2):189–208, 1967.

[150] C.-N. Yu and T. Joachims. Training structural SVMs with kernels using sampled cuts. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 794–802, 2008.

[151] C.-N. Yu and T. Joachims. Learning structural SVMs with latent variables. In Léon Bottou and Michael Littman, editors, *Proceedings of the 26th International Conference on Machine Learning*, pages 1169–1176, Montreal, June 2009. Omnipress.

[152] C.-N. 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.

[153] C.-N. Yu, T. Joachims, T. Elber, and J. Pillardy. Support vector training of protein alignment models. In *Proceeding of the International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 253–267, 2007.

[154] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 278. ACM, 2007.

[155] Y. Yue and T. Joachims. Predicting diverse subsets using structural SVMs. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1224–1231. ACM, 2008.

[156] A.L. Yuille and A. Rangarajan. The concave-convex procedure. *Neural Computation*, 15(4):915–936, 2003.

[157] Y. Zhang and J. Skolnick. TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic Acids Research*, 33(7):2302, 2005.

[158] A. Zien, U. Brefeld, and T. Scheffer. Transductive support vector machines for structured variables. In *Proceedings of the 24th International Conference on Machine Learning*, page 1190. ACM, 2007.