AN ABSTRACT OF THE DISSERTATION OF

Charles Parker for the degree of Doctor of Philosophy in Computer Science presented on August 27, 2007.

Title: Structured Gradient Boosting

Abstract approved: _____

Prasad Tadepalli

The goal of many machine learning problems can be formalized as the creation of a function that can properly classify an input vector, given a set of examples of that function. While this formalism has produced a number of success stories, there are notable situations in which it fails. One such situation arises when the class labels are composed of multiple variables, each of which may be correlated with all or part of the input or output vectors. Such problems, known as structured prediction problems, are common in the fields of information retrieval, computational linguistics, and computer vision, among others. In this dissertation, I will discuss structured prediction problems and some of the previous approaches to solving them. I will then present a new algorithm, *structured gradient boosting*, that combines strong points of previous approaches while retaining their generality. More specifically, the algorithm will combine some of the notions of margin maximization present in support vector methods

with the speed and flexibility of the structured perceptron algorithm. Finally, I will show a number of novel ways in which this algorithm can be applied effectively, highlighting applications in learning by demonstration and music information retrieval.

Structured Gradient Boosting

by

Charles Parker

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented August 27, 2007
Commencement June 2008

Doctor of Philosophy dissertation of Charles Parker presented on August 27, 2007.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electric Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Charles Parker, Author

ACKNOWLEDGEMENTS

Many thanks go to the faculty of the School of Electrical Engineering and Computer Science, who have not only provided useful insight and guidance as this work has progressed, but also given me the freedom to work as I chose. Dr. Prasad Tadepalli and Dr. Alan Fern deserve special recognition in this regard, as I could not have hoped for two more exceptional scientists to serves as advisers, collaborators, and advocates during my tenure at Oregon State University.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

DEDICATION

To my family.

Chapter 1 – Introduction

The field of machine learning has made great strides in solving many difficult machine learning problems by framing them as classification tasks. In classification, a learning algorithm is given a training set comprised of several vectors of numbers, each with an associated class label. The goal of learning is to find a function that will map any vector of numbers drawn from the same distribution to the correct class. This type of learning has produced significant results in cancer diagnosis [35], handwritten character recognition [53], and autonomous vehicle control [71] among many other areas.

As an instructive example, consider vectors that represent a person according to their height and weight, and classifying these people according to their sex. It is reasonable to assume, after a few training examples (155 cm, 55 kg, female; 185 cm, 80 kg, male, etc.) that a person would be able to make a reasonable[1] guess as to the sex of a person based on their height and weight.

In spite of the vast progress made, however, there are more problems than ever that seem immune to classical machine learning approaches. An example of this problem is the so-called *structured prediction* problem. In this formalism, the goal is still to predict the output "class label" as a function of the input vector. However, the shape of both the input and the class label is unrestricted. Both the input

[1] By "reasonable", I mean something substantially better than a one in two chance.

vector and the output label can be sequences, trees, bags, or have any structure imaginable. Typically, both input and output are represented as some collection of variables, with each variable in the output possibly having correlations with all or part of the input vector as well as other variables in the output structure.

Consider the problem of labeling an English sentence (input) with its proper parts of speech (output). In the sentence, "He is a stranger.", the word "stranger" is clearly a noun. However, adding a few words on the end, "He is a stranger man than I." results in the same word being an adjective. In this case, we see that one of the output variables (the part of speech of stranger) is correlated to several of the input variables (the word "stranger" as well as the words before and after it) and also possibly some of the other output variables (knowing that "stranger" is followed by a noun would point to its being an adjective).

Methods for solving these types of problems, such as maximum margin Markov networks [91] and structural support vector machines [93], are slowly emerging in the literature, with each new algorithm having its own strong points and weaknesses. Most recently, researchers in the community have begun to explore in depth the connections between existing algorithms, as well as ways of extending them to other types of problems. Given the recent advances, it is reasonable to assume that structured prediction will be in the literature for some time in the future.

1.1 Contributions

In this dissertation, I will examine some of the methods and applications in this new subfield of machine learning. I will first discuss structured prediction and try to establish what it is and what it is not. I will then broadly review some of the current methods of structured prediction and examine some others in more detail, seeking an algorithm that solves the most general version of the structured prediction problem as correctly and as quickly as possible.

The first contribution of this dissertation is the presentation of an algorithm, *structured gradient boosting*, that attempts to satisfy these algorithmic goals. This algorithm attempts to combine the best of two well-known algorithms, *structured perceptron* and *structural support vector machines* into an algorithm that is faster and more scalable than the latter, but uses a notion of margin and incorporates classification loss into its derivation, unlike the former. I then give results showing that this algorithm performs well in two simple applications.

The second contribution is to highlight the flexibility of this algorithm and push the boundaries of structured prediction in general. I show that, by learning classifiers to represent the gradient steps of the algorithm, I can effectively use the algorithm to do feature construction in the sequence alignment and retrieval domain. Learning these classifiers eliminates the need to define a distinct "character alphabet" and at the same time reduces the amount of training data required for accurate alignment and retrieval. I will also show that, with only a slight variation,

this algorithm can be used to tune the retrieval process to higher efficiency, with only small losses in accuracy.

1.2 Organization of this Dissertation

The organization of the rest of this dissertation is as follows:

In Chapter 2, I define the structured prediction problem and discuss how it differs from previous problems in machine learning. I discuss some methods that solve a subset of these problems, and then discuss a general form of the problem and methods that address this form. After discussing some of the advantages and shortcomings of some of these approaches, I introduce structured gradient boosting. I then show how this algorithm is able to combine some of the beneficial features of two of the most popular structured prediction algorithms. Finally, I show how structured gradient boosting can be applied to multi-class classification and give some results on common datasets.

Chapter 3 shows how structured gradient boosting can be applied to a simple instance of a structured prediction problem in learning by demonstration. Specifically, I will use structured prediction to develop a model that distinguishes good plans from bad plans, given some demonstration plans that are treated as near-optimal. I then show promising results on a planning problem in a real-time strategy game.

In Chapter 4, I begin with a literature review of my primary motivating application, sequence alignment and retrieval. In this chapter, I review some of the

methods of doing sequence alignment and retrieval. I then show that the success of all such methods hinges primarily on a function that calculates the edit distance between one character and another. With this in place, I review methods for learning this edit distance function and find that all current methods have deficiencies that cannot be remedied without substantial modification.

I then attempt to use structured gradient boosting to remedy some of these deficiencies. Here, I show how my algorithm can be used without substantial modification to solve the problem more quickly than the state of the art. In addition, my results show that using classifiers to represent the gradient steps of the algorithm results in a substantial representational benefit. This representation eliminates the need for a human expert to define an *ad-hoc* set of features, instead relying on algorithm to induce features from the training data. I show experimental results in a synthetic domain, comparing the results with other learning methods.

Chapter 5 introduces the *query-by-humming* domain and surveys some of the literature concerning this difficult problem in music information retrieval. I then outline my approach and show that structured gradient boosting is able to learn edit distances at least as well as competing approaches. It also possesses an ability to incorporate prior knowledge into the learning process, an attribute that some other discriminative approaches lack.

Going beyond sequence retrieval accuracy, I also show how structure gradient boosting can be used to improve retrieval efficiency in Chapter 6. I begin by showing how any distance function can be made to satisfy the triangle inequality through a simple mathematical transformation. Armed with this, I show how

retrieval efficiency can be improved through the use of *metric access methods*. Finally, I show how a distance function can be modified to improve the performance of a given metric access data structure using structured gradient boosting. I again show results in the synthetic and query-by-humming domains.

The final chapter, Chapter 7, concludes the dissertation. I summarize the contributions and give some thoughts on future direction for structured prediction and for structured gradient boosting in particular.

Chapter 2 – Structured Prediction: Problems and Approaches

In this chapter I will define, both formally and informally, the structured prediction problem. I will then review some previous approaches to this problem and examine their strong and weak points. Finally, I will outline an algorithm that combines some of the strong points of two of the most general algorithms reviewed, and show how it can be applied to multi-class classification.

2.1 Why Structured Prediction?

In the past 50 years, statistical machine learning has revolutionized both the field of artificial intelligence and many of the things we do every day. Computers are now able to perform tasks that would have never been possible without the learning algorithms developed in this community. We now have machines that are able to sort mail [53], diagnose cancer [35], control airplanes [71], recognize speech [72], and detect credit card fraud [12], among hundreds of other applications.

As stated in Chapter 1, many such problems are solved by reducing them to a *classification* or *regression* task [77]. These problems can be formalized as a tuple $\{\mathcal{X}, \mathcal{Y}\}$ where $\mathcal{X}$ is the *input domain* and $\mathcal{Y}$ is the *output domain*. We seek a function $F : \mathcal{X} \mapsto \mathcal{Y}$ that maps all inputs $\mathbf{x} \in \mathcal{X}$ to the correct output, or *class label* $y \in \mathcal{Y}$. The input domain for a given problem is, most generally, a vector

of real numbers[1] of some fixed size d so that $\mathcal{X} = \Re^d$. If $\mathcal{Y} = \Re$, then this is a *regression* problem. If $\mathcal{Y} = \{-1, 1\}$, then this is a *binary classification* problem. If $\mathcal{Y}$ is some reasonably small set, this is a *multi-class classification* problem.

To construct F, we are given a *training set* $\mathcal{T}$ of the form:

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}. \tag{2.1}$$

With these *training examples* of input vectors mapped to correct output values, the algorithm attempts to learn an F that will predict the correct ouput for parts of the input space unobserved in the training data. There is a wealth of methods for learning such functions, ranging from the venerable *perceptron* algorithm [76], to more recent methods such as Bayesian networks [34] and support vector machines [17].

To put this more concretely, suppose we have data on several past cancer patients. As part of this data, we have several numeric values about the patient's health (age, weight, blood counts, etc.) and also a class label $\in \{-1, 1\}$ indicating whether or not the patient's cancer is malignant. Given a training set of many such patients, the goal is to learn a function that, given the same set of numeric values about an unknown patient, will predict whether or not the patient's cancer is malignant.

For many problems, this formalism is adequate. However, there are many important problems for which this formalism appears appropriate, but only on

[1]This representation generalizes easily to cases where there are nominally-valued elements in the vector, but such a discussion is outside of the scope of this dissertation. For a discussion, see [77].

the surface. Consider the problem of mapping an English sentence to its proper French translation [50]. On the surface, we have the usual definitions of an input and output space where $\mathcal{X}$ is the set of all possible English sentences and $\mathcal{Y}$ is the set of all possible French sentences. If we have a training set English sentences paired with the correct French translation, we may assume that existing learning algorithms will be able to learn the appropriate mapping.

Unfortunately, this assumption is incorrect. Classical statistical learning algorithms rely on the number of classes being relatively small, or the class label being real-valued. Essentially, this is because these algorithms all rely on *a priori* knowledge about the domain of the output value. In the problem described above, the number of class labels is infinite and many of them (possible French sentences) are not present in the training set. Classical statistical learning methods do not typically predict classes for which they have no training examples[2].

All, however, is not lost. While an exponential number of classes is clearly a cause for concern, it does not seem to be so in this case. For example, although there are exponentially many parse trees for a given sentence, many of them are very much the same as many others, while many others still are very different. In other words, the possible classes for the problem, the output space, appears to have structure itself. It seems possible that a learning algorithm could leverage this structure to predict the correct class, or at least a class that is in some sense "close" to correct.

[2]Excepting the real-valued output space (regression).

2.2 Structured Prediction

In the last five to ten years, a new type of statistical learning, known as *structured prediction* has emerged in the literature. The goal of structured prediction is to bring statistical learning to bear against problems having an output space that is structured but exponential in size. Structured prediction can be seen as a generalization of the prediction problem above. In the case of structured prediction, $\mathcal{X}$ and $\mathcal{Y}$ are still the input and output domains, respectively, but the nature of these domains is now unrestricted. It may be a space of sequences, trees, images, or a combination of these. The goal is still the same: To find an $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps each possible input structure $\mathbf{x} \in \mathcal{X}$ to the correct output $\mathbf{y} \in \mathcal{Y}$.

A number of methods for doing this have appeared in the literature in recent years, including *structured perceptron* [14], *searn* [19], *maximum margin Markov networks* [91], *structural support vector machines* [93], *maximum entropy Markov models* [55], and *conditional random fields* [47]. While the methods for doing this vary considerably between these algorithms, there are some common threads. It is these that I will focus on in my definition of structured prediction below.

2.2.1 Some Related Work

Before formally defining the general structured prediction problem, I will briefly examine two of the methods described above. These methods solve a subset of structured prediction problems and have served as important springboards to more general solutions.

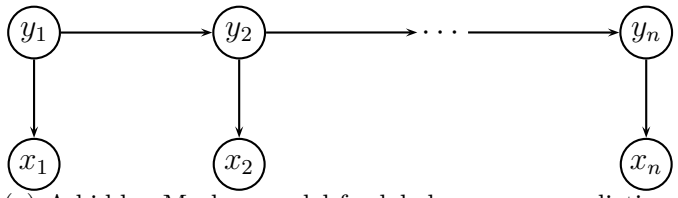
2.2.1.1 Conditional Random Fields

Conditional random fields are a conditionally-trained alternative to traditional graphical models. Graphical models (which will be explored in later in this chapter and in Chapter 4) are useful in applications where prior knowledge about the domain can be used to construct a more accurate generative model. Unfortunately, these models are often used as "off the shelf" prediction methods for problems such as label sequence learning. This is unfortunate because these models expend a great deal of effort modeling the joint probability distribution $P(\mathbf{x}, \mathbf{y})$, which includes interactions within the observed structure $\mathbf{x}$. Obviously, there is no need to model these interactions because $\mathbf{x}$ is completely observed at classification time.

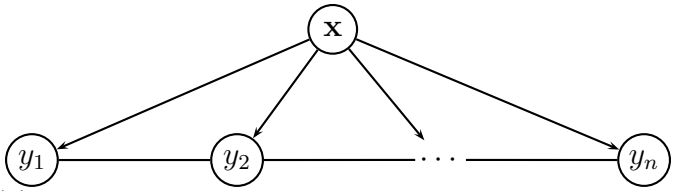
Conditional random fields use a *feature-based* approach to representing the clique potentials of a given graphical model. That is, for a given clique c in a graphical model, the clique potentials are represented as an exponential function of some parameterized combination of features defined over that clique. Suppose $f(c)$ gives the feature vector for clique c and $\mathbf{w}$ is a vector of the parameters. The clique potential, $\psi(c)$ of the clique is expressed as:

$$\psi(c) = \exp(\langle \mathbf{w}, f(c) \rangle) \tag{2.2}$$

Figure 2.1 shows a conditional random field designed to predict a *label sequence*, such as parts of speech for a sentence, given a sentence $\mathbf{x}$. As we can see in Figure 2.1(b), the conditional random field makes no assumptions about the structure of $\mathbf{x}$. Suppose the function f gives the feature values for all cliques of size 2 in the



(a) A hidden Markov model for label sequence prediction.



(b) A conditional random field for label sequence prediction.

Figure 2.1: The difference in modeling assumptions between hidden Markov models for label sequence prediction and conditional random fields for the same task.

above graph and the function g gives feature values for all cliques of size 3. Also suppose that the parameters for these potentials are given by $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ respectively. Then, the clique potentials for the graph in Figure 2.1(b) are:

$$\psi(\mathbf{x}, y_i) = \exp(\langle \boldsymbol{\lambda}, f(\mathbf{x}, y_i) \rangle) \tag{2.3}$$

$$\psi(\mathbf{x}, y_i, y_{i+1}) = \exp(\langle \boldsymbol{\mu}, g(\mathbf{x}, y_i, y_{i+1}) \rangle) \tag{2.4}$$

The feature-based representation of the clique potentials is important for two reasons: First, the potentials can be conditioned on arbitrary attributes of the input structure $\mathbf{x}$. For example, a feature in the part-of-speech tagging example could be "the feature value is 1 if the sentence begins with the word 'What' and

ends with a question mark. Otherwise the value is 0". Second, it gives a sparsely parameterized representation for the clique potential functions, which makes inference less complex and demands less data for training [35].

With the clique potentials defined as such, we can compute the conditional probability of a given label sequence $\mathbf{y}$ as we would in a standard graphical model:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{i}^{n-1} \psi(\mathbf{x}, y_i, y_{i+1}) \prod_{i}^{n} \psi(\mathbf{x}, y_i) \tag{2.5}$$

where Z normalizes the product to be a proper distribution. Training can then take place using maximum likelihood estimation as in any graphical model. Because this model is undirected, we must use an iterative method like *iterative scaling* [39] to find the maximum likelihood solution given the training data.

2.2.1.2 Max-Margin Markov Networks

The max-margin Markov Network (or M^3N for the geek-chic) gets more to the heart of the problem. First, note that we can form a single feature vector, Ψ that concatenates the feature vectors from each clique:

$$\Psi(\mathbf{x}, y_i, y_{i+1}) = (f(\mathbf{x}, y_i), g(\mathbf{x}, y_i, y_{i+1})) \tag{2.6}$$

If we assume that the potentials are the same from end to end on the label sequence model, we can define a *joint feature vector* over the input and output structures:

$$\Psi(\mathbf{x}, \mathbf{y}) = \sum_{i}^{n} (f(\mathbf{x}, y_i), g(\mathbf{x}, y_i, y_{i+1})) \tag{2.7}$$

with the convention that $g(\mathbf{x}, y_n, y_{n+1}) = 0$. Now suppose that we group the parameters of the potential functions into a single vector as well, so that $\mathbf{w} = (\boldsymbol{\lambda}, \boldsymbol{\mu})$. With this, we have:

$$P(\mathbf{y}|\mathbf{x}) \propto \exp(\langle \Psi(\mathbf{x}, \mathbf{y}), \mathbf{w} \rangle) \tag{2.8}$$

Obviously, the $\mathbf{y}$ in which we are interested when performing inference is the one that maximizes the conditional probability given the input structure and parameters of the distribution:

$$\operatorname*{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \operatorname*{argmax}_{\mathbf{y}} \exp(\langle \Psi(\mathbf{x}, \mathbf{y}), \mathbf{w} \rangle) \tag{2.9}$$

$$= \operatorname*{argmax}_{\mathbf{y}} \langle \Psi(\mathbf{x}, \mathbf{y}), \mathbf{w} \rangle \tag{2.10}$$

And so, crucially, the optimal parameters for the conditional random field turn out to be the same as those for the best linear classifier given the appropriate feature vectors. Given a training set, we can then formulate the problem of learning the parameters $\mathbf{w}$ as a constrained optimization problem as is done in support vector machines [17]. In Section 2.3.1, we explore this optimization problem in greater detail. The problem with this formulation (in both Section 2.3.1 and in

max-margin Markov networks) is that the resulting optimization has a number of constraints that is $O(|\mathcal{Y}|)$. In a prediction problem such as this one, where $\mathcal{Y}$ is all possible sequences of labels, this is an intractably high number.

The approach taken to solve this optimization in max-margin Markov networks is to use the structure of the graph to obtain an efficient solution. Essentially, because the joint feature vector is decomposable across repeating cliques of the model, the resulting optimization is decomposable as well, and is efficiently solvable.

2.2.1.3 Discussion

Conditional random fields do away with modeling the entire joint distribution in problems where it is not necessary. Using feature functions to describe the clique potentials results in a graphical model that is both more tractable for learning purposes and more meaningful semantically. Max-margin Markov networks take this idea one step further, learning the parameters through the solution of a constrained optimization and gaining generalization guarantees in the process.

Unfortunately, the thing that makes the learning tractable in conditional random fields and the optimization problem solvable in max-margin Markov networks turns out to be the most limiting factor of these methods. In particular, the repeating structure of the graphs contributes greatly to the efficiency of the computations involved, but it also forces us to use features that are parameterized by the variables in each clique and no others. In addition, while graphical models provide a useful way of encoding prior knowledge into the learning process, they also require

an expert of some sort to properly specify which variables do and do not interact as well as what those interactions are.

I seek a more general solution to the structured prediction problem, where the structure of the underlying model can be arbitrary. That is, I wish for the features of the model to be any arbitrary combination of attributes of the input and output structure. To this end, I give the formal and more general problem definition below:

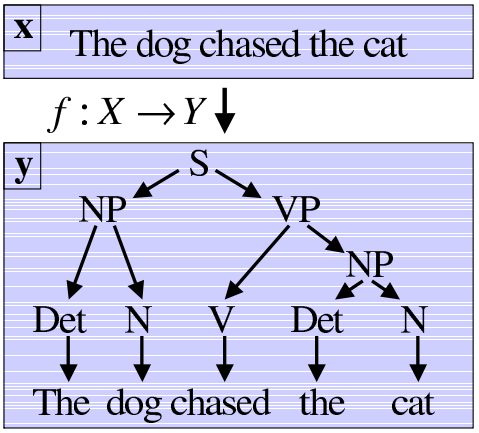
2.2.2 Problem Definition

In this work, a *structured prediction problem* is a tuple $\{\mathcal{X}, \mathcal{Y}, \Psi, \Delta\}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input and output domains for the problem. These domains can be comprised of values or structures of any type.

The function $\Psi : \mathcal{X} \times \mathcal{Y} \mapsto \Re^d$ is a *joint feature vector extractor* that takes a member of the input and output spaces and returns a vector of real values. Semantically, the elements of this vector are a list of features that result from classifying the input element with the given output element. For example, in machine translation [50], this could be the number of times a matching pair of words or phrases (such as, [thirty-five, trente-cinq]) appears in the translated pair. An intermediate goal, then, is to construct Ψ so that it extracts features relevant to determining whether the input/output pair passed to it is one of high quality.

A second function, $\Delta : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \Re$, is the misclassification cost between two classes for a given input instance. This loss function should be, semantically,

a measure of *how much less correct* the first class label is than the second label. For example, suppose we are given the input sentence in English "He does not drive the car every day" and the correct translation in French, "Il ne conduit pas la voiture tous les jours". The classification loss between the correct label, and for the incorrect label, "Il ne pas conduit la voiture tous les jours" (correct words but improper grammar), would be substantially lower than the difference in loss between the correct label and "Tu ne chantes pas une chanson dans la rue" ("You do not sing a song in the street"[3]).



$$\Psi(\mathbf{x},\mathbf{y}) = \begin{pmatrix} 1 & S \rightarrow NP\,VP \\ 0 & S \rightarrow NP \\ 2 & NP \rightarrow Det\ N \\ 1 & VP \rightarrow V\ NP \\ \vdots & \\ 0 & Det \rightarrow dog \\ 2 & Det \rightarrow the \\ 1 & N \rightarrow dog \\ 1 & V \rightarrow chased \\ 1 & N \rightarrow cat \end{pmatrix}$$

Figure 2.2: An English sentence and its correct parse tree, with the associated Ψ vector.

Another structured prediction domain is shown in pictorial form in Figure 2.2, taken from [93]. We wish to take an English sentence as the input domain $\mathcal{X}$ and output its proper *parse tree* as the output domain $\mathcal{Y}$. The Ψ vector gives counts of the various *productions* that occur in the parse tree. This is a complex problem

[3]This may or may not be true.

in natural language processing [13] that I will not examine in detail here. It is enough to note the complexity of the prediction task and how it may be fit into the formalism I have defined.

Note that this is not to suggest that all other problems are *unstructured*. For example, credit card fraud detection and stock market prediction both have temporal (and even geo-spatial) aspects that certainly speak of the problem being structured in some sense. That is, if we knew the sequence of credit card transactions and their labels (fraud/not fraud) leading up to the current transaction, then we would be in a better position to determine the validity of that transaction than if we had no knowledge of these labels. One way to accomplish this is to treat the labels leading up to the current label as separate prediction problems, the predictions from each one propagating forward until reaching the current label (the so-called *sliding window* method [21]). In this way, the answers from previous problems can become features for future predictions.

In some cases, variables that are of interest but not part of the input are treated as *hidden variables*. As above, we attempt to predict these values even though they are not of primary interest, and use those predictions to help predict the output variable. Usually, this involves some form of iterating between guessing the value of the hidden variables, then re-estimating the model parameters based on these guesses [20]. Although situations involve prediction problems that have structure, we are here primarily interested in problems where the *prediction itself is structured*. That is, problems with more than one output variable. Note before moving

on, however, that wherever there are multiple values to be predicted, structured prediction methods may succeed where others have failed.

Note that the definition above admits problems such as classification and regression that have only a single output variable. Is there, then, a proper definition of a structured prediction problem? Some recent informal discussions[4] have suggested the following conditions to differentiate structured prediction problems from standard prediction, as well as differentiating structured prediction algorithms from standard prediction algorithms:

1. The output domain $\mathcal{Y}$ is a *set of variables to be predicted*, which may be variable in length.

2. The feature function Ψ is *not* decomposable over the output variables.

3. The loss function Δ is *not* decomposable over the output variables.

If the first condition is satisfied, but neither of the other two are, the problem can be solved by multiple independent classifiers [70]. This means that the "structure" of the prediction is simply a concatenation of ordinary prediction problems and should not be included in any strict definition of structured prediction.

The other two conditions are more elusive. In some sense, the second condition implies structure because the features imply correlations between the output variables. However, such features could be mistakenly introduced for a problem

[4]In particular, the machine learning blog run by Hal Daumé at `http://nlpers.blogspot.com/` has much lively discussion on this very topic.

with no such correlations. These features would clearly be irrelevant, but prediction problems are rife with such features and any good algorithm is robust against them. In this case, the first two conditions hold, and yet the problem is still clearly unstructured because the output variables could still be predicted independently with success. Because the correlating features are irrelevant, their presence does not imply structure. It seems, then, that the ability to incorporate features that express correlations between the output variables is more an attribute of the *algorithm* than of the problem itself.

One way to guarantee the relevance of the correlating features is the third condition. If the loss is not decomposable over the output variables, than we require correlating features to minimize this loss function properly. Note that this definition, while adequately restrictive, is almost certainly too strict. Label sequence prediction under hamming loss, for example, does not satisfy this third condition, yet many would consider it to be a structured prediction task.

In summary, it appears there are several main issues we confront when attempting to define structured prediction. First, there is a question of what exactly a structured prediction problem is, to which I have responded that it is a prediction problem with multiple, possibly correlated output variables. Second, there is a question of what exactly is a structured prediction algorithm, to which I have responded that it is an algorithm that can incorporate features that correlate the output variables. Third, there is a question of when it is useful or necessary to apply a structured prediction algorithm. One way that has been proposed is related to the decomposition of the loss function, but this only seems to capture a subset

of all instances in which structured prediction algorithms are useful. Other, more subtle forms of correlation may still benefit from a structured approach.

There are some indications, again in informal discussion, that these conditions do not stand up to rigorous formal scrutiny. Nonetheless, they do provide some interesting intuitions about structured prediction. For the remainder of this work, I will use the broad definition given at the beginning of the section. Under this definition, structured prediction is a generalization of standard prediction. Accordingly, the algorithms discussed in detail in what follows will all readily specialize to classification and regression.

2.2.3 Some Other Structured Prediction Algorithms

The over-arching theme of many of the structured prediction algorithms not discussed thus far [93, 67, 14, 19] is an iterative structure that proceeds roughly as follows:

1. Use the current model to choose the best incorrect class label for each training example. That is, choose the best class label that is *not* the one given in the training set.

2. Compare these predicted solutions with the correct class label for each training example (given in the training set) and adjust the model so it favors the correct solutions rather than the errant predictions.

3. Repeat until convergence.

Note that in step one of the above algorithm, the inference routine has been embedded into the learning algorithm. Inference in many of the methods above takes the form of a series of inner products between the feature vector, given by Ψ, and a *weight vector*, $\mathbf{w}$, so that each element of $\mathbf{w}$ is a weight for a corresponding element in Ψ. The inner product, $\langle \Psi(\mathbf{x}, \mathbf{y}), \mathbf{w} \rangle$, computes a number that can be treated as a quantitative measure of how appropriate it is to apply the label $\mathbf{y}$ to the input $\mathbf{x}$. To complete the inference routine, the algorithm chooses the best of these labels according to this inner product, so the best label is:

$$\underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}} \langle \Psi(\mathbf{x}, \mathbf{y}), \mathbf{w} \rangle \tag{2.11}$$

There are two problems here: First, computing the argmax above is almost certainly non-trivial. As stated before, the number of classes is often infinite or at least exponential in structured prediction problems, and so the argmax cannot be found efficiently by simply trying each possible $\mathbf{y} \in \mathcal{Y}$. Fortunately, there are a variety of ways to make inference tractable in these algorithms. The method of doing so is often specific to both the learning algorithm and the domain. For parse tree construction (an important structured prediction problem in natural language processing), this can be done using the CKY algorithm [13]. In sequence matching and alignment (which I will discuss later), there is the Smith-Waterman algorithm [88], or, for graphical models, the Viterbi algorithm [96].

Secondly, the algorithm sketch shown at the beginning of the section requires that we find the best *incorrect* class label for a given input[5]. Throughout what

[5]Being able to find the best class label outright will, however, be useful at performance time.

follows, I will designate the best incorrect class label using a "hat" ($\widehat{\cdot}$). To illustrate, suppose $\mathbf{y}_i$ is the correct class label for a given input $\mathbf{x}_i$. We can compute $\widehat{\mathbf{y}}_i$, the best incorrect class label for $\mathbf{x}_i$, with just a little change to the inference routine above:

$$\widehat{\mathbf{y}}_i = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}, \mathbf{y} \neq \mathbf{y}_i} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle \tag{2.12}$$

All that remains in the learning algorithm is step two: the modification of the model to give higher inner products to the correct answers. This is the essential difference between all of the proposed algorithms. In conditional random fields [47], the model is trained to maximize the conditional probability of the best class label for each input. In structural support vector machines [93] and the structured perceptron [14], the model is trained to maximize the margin between the best class label and the incorrect class labels.

2.3 Structural Support Vector Machines and Structured Perceptron

I have now defined the structured prediction problem and visited briefly some of the attempts to solve it. I will now choose two of these methods to examine in more detail and discuss their advantages and disadvantages.

2.3.1 SVM-Struct

The structural support vector machine (SVM-Struct) maximizes the margin of the training data by solving iteratively harder quadratic programming problems. Essentially, this algorithm aims to solve the the following set of constraints:

$$\forall i \in \{1, \ldots, n\}, \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i : \ \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle > 0 \tag{2.13}$$

This assumes linear separability of the training examples in the feature space. If the data are linearly separable, then there will typically be more than one $\mathbf{w}$ that satisfies the constraints in Equation 2.13. To make the problem well-posed, the norm of $\mathbf{w}$ is constrained so that $||\mathbf{w}|| = 1$ and choose the $\mathbf{w}$ that maximizes the distance between all correct labels of the training data and their closest incorrect competitors. This leads to the margin maximization problem:

$$\max_{m, \mathbf{w}: ||\mathbf{w}||=1} m \tag{2.14}$$

$$\text{s.t. } , \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i : \ \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle > m \tag{2.15}$$

Performing the usual transformations relating to support vector machines [17] leads to the following quadratic program:

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 \tag{2.16}$$

$$\text{s.t. } \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i : \ \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle \geq 1 \tag{2.17}$$

If the data are not linearly separable in the feature space, then *slack variables* ξ_i can be introduced to allow for margin violations, and we can optimize a soft-margin criterion instead:

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{n} \sum_i \xi_i \tag{2.18}$$

$$\text{s.t. } \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i : \ \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle \geq 1 - \xi_i, \xi_i > 0 \tag{2.19}$$

Finally, misclassification cost functions other than the standard zero-one loss seen above can be employed. More specifically, one option is to *rescale the margin* according to the loss function [92] so that margin violations are additively "shifted" by the amount of the loss:

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{n} \sum_i \xi_i \tag{2.20}$$

$$\text{s.t. } \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i : \ \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle \geq \Delta(\mathbf{x}, \mathbf{y}, \mathbf{y}_i) - \xi_i, \xi_i > 0 \tag{2.21}$$

This has the unpleasant property that the solution is not constant under a scalar multiple of the misclassification cost function. In other words the optimal solution for $\mathbf{w}$ may be different with a loss function that is semantically congruent. Another idea, is to *rescale the slack variables*, so that the margin violations are scaled by the loss:

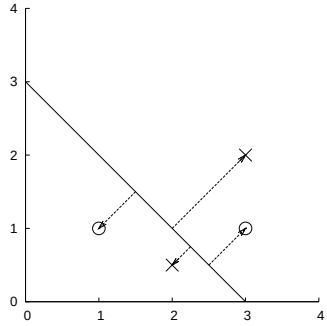
$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{n}\sum_i \xi_i \tag{2.22}$$

$$\text{s.t. } \forall i, \forall \mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i : \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle \geq 1 - \frac{\xi_i}{\Delta(\mathbf{x}, \mathbf{y}, \mathbf{y}_i)}, \xi_i > 0 \tag{2.23}$$

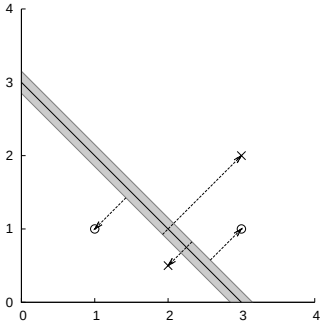
This formulation is consistent under scalar multiples of the misclassification cost. Figure 2.3 shows how the slack variables are effected by margin and slack rescaling. Points marked with "o" are on the correct side of the decision boundary. Those marked with "x" are on the incorrect side.

With this in place, all that is left is the solution of this quadratic program. Unfortunately, $|\mathcal{Y}|$ may be extremely large, especially if $|\mathcal{Y}|$ is a product space such as the number of possible French sentences. Because the number of constraints in Equation 2.23 is $n|\mathcal{Y}| - n$, solving this optimization is not feasible.

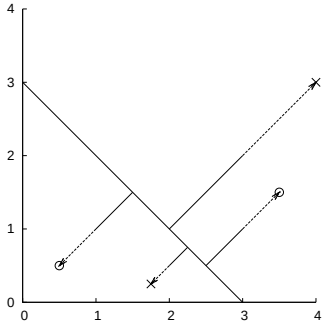
The approach taken by structural support vector machines [93, 92] is to start with an empty set of constraints and iteratively add the *most violated constraint* to



(a) Original space before rescaling.



(b) Margin rescaling.



(c) Slack rescaling.

Figure 2.3: The effects of margin and slack rescaling on distances from the decision boundary.

the constraint set, given the learned model, and resolve the resulting optimization[6]. This algorithm is proven to converge to some ϵ of the optimal solution by adding a number of constraints that is only polynomial in the input size, thus making an approximate solution to this problem a possibility.

2.3.2 Structured Perceptron

The structured perceptron algorithm [14] is a generalization of the perceptron algorithm to structured prediction. At each iteration, the algorithm choses best incorrect solution for each training example and uses this to update the current $\mathbf{w}$. Unlike structural support vector machines, however, the weights are simply updated according to the difference in feature vectors between the correct and incorrect answers. The intuition is to push the weights of features with greater values in the correct class labeling higher, and vice-versa with features having greater value in the incorrect labeling:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(\Psi(\mathbf{x}_i, \mathbf{y}_i) - \Psi(\mathbf{x}_i, \widehat{\mathbf{y}}_i)) \tag{2.24}$$

The factor α is a learning rate parameter that controls the size of the update.

Exactly how often this update happens is deliberately vague. There are several versions of the algorithm. Sometimes, the updates are summed over the entire training set before they are applied, and each iteration of the algorithm constitutes

[6]To find the most violated constraint is to find $\widehat{\mathbf{y}}_i$ for each $\mathbf{x}_i$ and choose the one for which the score is higher for $\widehat{\mathbf{y}}_i$ than $\mathbf{y}_i$ by the greatest amount.

a pass over the entire training set [14]. Another version of the algorithm updates greedily after each training example.

A more clever formulation, the *voted perceptron* [28] makes updates only when the current model makes a mistake in the training set (that is, when the class predicted by the model is incorrect). When a mistake is made, the number of examples correctly classified before the mistake serves as the weight for the model. A new model is then created by updating the current model, the current model is stored, and the process continues with the new model as the current model. The number of models stored, then, is equal to the number of mistakes made during training. The resulting ensemble classifier has some of the nice theoretical properties of *adaboost* [79].

2.3.3 Discussion

I have discussed two methods of structured prediction. Let us now examine their relative strengths and weaknesses. Both algorithms try to optimize a notion of margin and both can be kernelized [95] according to the standard methods [17].

SVM-struct, however, has some obvious advantages:

- **Precision of solution:** SVM-struct exactly solves a series of optimization problems to reach its final solution. The gradient-based optimization of the structured perceptron is not even tailored specifically to the margin - it gives only a simple push toward the correct solution with little regard for the performance of the current model.

- **Robust theoretical guarantees:** The solution of SVM-struct is guaranteed to be precise within some epsilon of the exact solution after solving some polynomial number of quadratic programs. It also carries with it the benefits of the regularization of standard support vector machines, making the solutions provided by the algorithm reasonable even in infinite dimensional spaces (such as the space implied when using a radial basis kernel [17]). Kernels will be discussed later on in the chapter.

- **Theoretical incorporation of general misclassification cost functions:** Using either margin or slack rescaling, as mentioned earlier, a general misclassification cost function can be incorporated elegantly into the problem formulation. For the structured perceptron, there is no obvious, theoretically sound way to do this.

Structured perceptron, too, has strong points:

- **Speed and scalability:** SVM-struct solves a number of quadratic programs polynomial in the size of the training set. If these problems are complex, involving thousands [67] or even millions [50] of features, the algorithm will not be able to admit very much training data. The structured perceptron algorithm uses only simple dot products and vector subtraction as part of the learning algorithm, and so can handle large amounts of data.

- **No requirement of exact inference:** Finding the "most violated constraint" in the constraint set of Equation 2.23 is non-trivial. As stated before,

$|\mathcal{Y}|$ could be extremely large. In the absence of an exhaustive search, the alternatives are an analytical computation of $\widehat{\mathbf{y}}_i$ (which may or may not be possible) or a stochastic search of the space of possible classes for an approximate answer. SVM-struct does not admit the latter of these two solutions in theory, and a few recent results at workshops suggest that inexact inference is extremely detrimental to the performance of the algorithm. By contrast, there is a variant of structured perceptron [15] that is able to cope with inexact inference in the inner loop.

- **Ability to operate in on-line mode:** The flexibility of structured perceptron in its update structure enables it to do updates as data become available, resulting in a model that can be updated continually with little effort. In addition, the model can be initialized by an expert with domain knowledge beforehand and learning can proceed with this knowledge as a starting point. The performance of SVM-struct, however, again depends on solving a series of quadratic problems. Updating the model with new data must consider all previous training data to achieve theoretical consistency.

We see then, that both structured perceptron and SVM-struct have certain strengths and weaknesses. Next, I will attempt to go some way towards reconciling the two. Specifically, I will attempt to construct a perceptron-style algorithm that incorporates a definition of margin and a general misclassification cost function into its strategy.

2.4 Structured Gradient Boosting

In this section I present "Structured Gradient Boosting", a reformulation of the structured perceptron algorithm that admits general misclassification cost functions and is margin-based. To derive this algorithm, I will first create a notion of training loss based on the margin violation at each training example. I will then take the gradient of this proposed training loss to obtain the desired update. Following this, I will show how general misclassification cost functions can be incorporated into the update and how the algorithm can incorporate kernels.

I begin by defining a notion of margin for the structured prediction problem. Again suppose that we have some training example $\mathbf{x}_i$ with the correct label $\mathbf{y}_i$. I define $\widehat{\mathbf{y}}_i$ to be the best incorrect label for $\mathbf{x}_i$ according to the current model. That is,

$$\widehat{\mathbf{y}}_i = \underset{\mathbf{y}\in\mathcal{Y},\mathbf{y}\neq\mathbf{y}_i}{\operatorname{argmax}} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle \tag{2.25}$$

We can then define the margin to be the amount by which the model prefers $\mathbf{y}_i$ to $\widehat{\mathbf{y}}_i$ as a label for $\mathbf{x}_i$. Because the difference in feature vectors will appear so often in this section, I define the shorthand $\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i) = \Psi(\mathbf{x}_i, \mathbf{y}_i) - \Psi(\mathbf{x}_i, \widehat{\mathbf{y}}_i)$.

The margin m_i for a given training example is then:

$$m_i = \langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle \tag{2.26}$$

I continue by borrowing a margin-based formulation of training loss first used in "logitboost" [29] for standard binary classification learning. In this formulation, the training loss is a function of the margin at each training example:

$$L(\mathbf{x}, \mathbf{y}_i, \widehat{\mathbf{y}}_i) = \log(1 + e^{-m_i}). \tag{2.27}$$

This creates a training loss function that is zero-bounded and differentiable. Figure 2.4 shows the relationship of the loss to the margin under this function.

Taking the gradient of this function with respect to the parameters of the model (in this case, the vector $\mathbf{w}$), yields the appropriate direction for the gradient descent step, ∇L:

$$\nabla L = \frac{dL}{d\mathbf{w}} \tag{2.28}$$

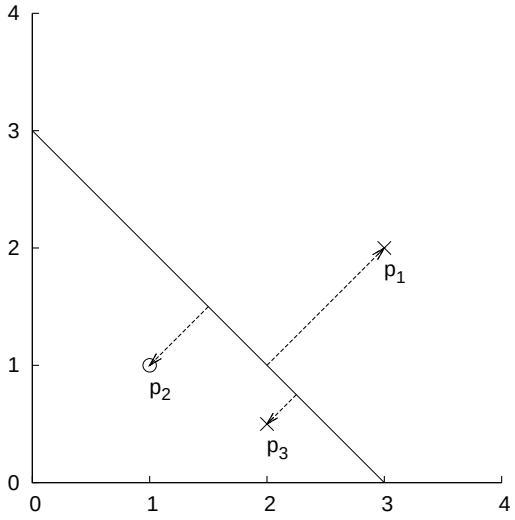
$$= \frac{\frac{d}{d\mathbf{w}}(1 + e^{-m_i})}{1 + e^{-m_i}} \tag{2.29}$$

$$= \frac{\frac{d}{d\mathbf{w}}(-m_i)e^{-m_i}}{1 + e^{-m_i}} \tag{2.30}$$

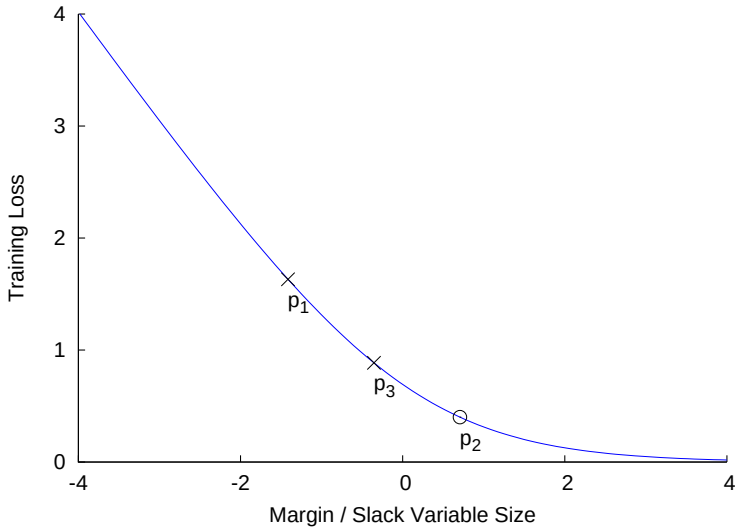
$$= \frac{\frac{d}{d\mathbf{w}}(-m_i)}{1 + e^{m_i}} \tag{2.31}$$

The last line is the most general construction of the update term for structured gradient boosting. Although I will focus on linear models[7] for the remainder of

[7]Linear in either feature space or kernel space.



(a) 2-d decision boundary with several points.



(b) Loss function as it related to the distance from the margin.

Figure 2.4: Relationship of the margin to the amount of training loss.

this thesis, we can note that the only assumption made here is that the margin is differentiable with respect to the model parameters. If the training loss function decreases monotonically to the global optimum and the learning rate is chosen appropriately, the algorithm will converge.

If the learning rate is α, then the gradient update at each iteration of the algorithm will be:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L \tag{2.32}$$

Specializing this update for the structured prediction case, we note that the margin is given by Equation 2.26, and so:

$$-\frac{dm_i}{d\mathbf{w}} = -\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i) \tag{2.33}$$

Substituting into Equation 2.31:

$$\nabla L = \frac{-\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)}{1 + \exp(\langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle)} \tag{2.34}$$

Recall the update term from the structured perceptron algorithm in Equation 2.24:

$$\Psi(\mathbf{x}_i, \mathbf{y}_i) - \Psi(\mathbf{x}_i, \widehat{\mathbf{y}}_i) \tag{2.35}$$

The structured gradient boosting update, then, ends up being roughly the same, except that the update is weighted by the margin violation at each training

example. This could also be viewed as a re-weighting of the training set at each iteration, as is done in *adaptive boosting* [79]. One could also view structured gradient boosting as a *self-tuning learning rate*, where the size of the gradient step is modified at the instance level based on the margin and the misclassification cost at each example.

2.4.1 Incorporation of Misclassification Cost Functions

Drawing from previous work, it is reasonably straightforward to incorporate misclassification cost into the update term. One approach, as shown for SVM-struct in Equation 2.21, is to rescale the margin [92]. That is, we widen the margin by the misclassification cost, so that the margin violations are additively shifted. This gives a training loss function of $L(\mathbf{x}, \mathbf{y}_i, \widehat{\mathbf{y}}_i) = \log(1 + e^{-m_i + \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)})$. After $\widehat{\mathbf{y}}_i$ is chosen, it can be treated it as a constant so that the derivative of the misclassification cost with respect to the model parameters is zero. With this, we can re-derive the update as follows:

$$\nabla L = \frac{dL}{d\mathbf{w}} \tag{2.36}$$

$$= \frac{\frac{d}{d\mathbf{w}}(1 + e^{-m_i + \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)})}{1 + e^{-m_i + \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)}} \tag{2.37}$$

$$= \frac{\frac{d}{d\mathbf{w}}(-m_i)e^{-m_i + \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)}}{1 + e^{-m_i + \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)}} \tag{2.38}$$

$$= \frac{\frac{d}{d\mathbf{w}}(-m_i)}{1 + e^{m_i - \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)}} \tag{2.39}$$

We notice that the inclusion of the misclassification cost has changed little in the derivation. The weight of the update has simply been modified by the misclassification cost at the example in question. We show the structured gradient boosting update for completeness:

$$\nabla L = \frac{-\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)}{1 + \exp(\langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle - \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i))} \tag{2.40}$$

Another alternative is again to rescale the slack variables [92], as shown for SVM-struct in Equation 2.21 so that the margin violations are multiplied by the misclassification cost. This gives a training loss function of $L(\mathbf{x}, \mathbf{y}_i, \widehat{\mathbf{y}}_i) = \log(1 + e^{-m_i \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)})$, and the derivation proceeds as follows:

$$\nabla L = \frac{dL}{d\mathbf{w}} \tag{2.41}$$

$$= \frac{\frac{d}{d\mathbf{w}}(1 + e^{-m_i \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)})}{1 + e^{-m_i \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)}} \tag{2.42}$$

$$= \frac{\frac{d}{d\mathbf{w}}(-m_i) e^{-m_i \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)}}{1 + e^{-m_i \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)}} \tag{2.43}$$

$$= \frac{\frac{d}{d\mathbf{w}}(-m_i \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i))}{1 + e^{m_i \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)}} \tag{2.44}$$

The structured gradient boosting update that results is:

$$\nabla L = \frac{-\Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i) \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)}{1 + \exp(\Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)[\langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle])} \tag{2.45}$$

Slack rescaling results again in only a scaling of the update term, but this time the misclassification cost has far more influence on the update. Specifically, for misclassification costs that are extremely large, the update will grow and shrink much more quickly as the example moves away from the margin on either side. By contrast, margin rescaling will only widen the margin boundary, while the rate of growth of the update remains the same as in the uniform training loss case.

We can achieve a happy medium between these two by *rescaling the training loss* according to the misclassification cost. This results in a training loss function of $L(\mathbf{x}, \mathbf{y}_i, \widehat{\mathbf{y}}_i) = \Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)(\log(1 + e^{-m_i}))$. Deriving the update:

$$\nabla L = \frac{dL}{d\mathbf{w}} \tag{2.46}$$

$$= \frac{\frac{d}{d\mathbf{w}}\Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)(1 + e^{-m_i})}{1 + e^{-m_i}} \tag{2.47}$$

$$= \frac{\Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)\frac{d}{d\mathbf{w}}(-m_i)e^{-m_i}}{1 + e^{-m_i}} \tag{2.48}$$

$$= \frac{\Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)\frac{d}{d\mathbf{w}}(-m_i)}{1 + e^{m_i}} \tag{2.49}$$

Specializing for structured prediction gives us:

$$\nabla L = \frac{-\Delta(\mathbf{x}, \widehat{\mathbf{y}}_i, \mathbf{y}_i)\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)}{1 + \exp(\langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w} \rangle)} \tag{2.50}$$

In this case, the update size still scales with the misclassification cost, but not quite as much as in the slack rescaling case. Exactly which version of training loss one wishes to optimize is certainly domain specific and the choice could result in dramatically different levels of performance.

2.4.2 Update Strategy

There are several different update strategies for the structured perceptron algorithm, each of which has its own possibilities and drawbacks, and each of which may be applied to structured gradient boosting. In the original update [14] the update terms are summed over the entire training set. This mega-update is ap-

plied, and the process is iterated. This formulation allows for some theoretical guarantees when there is exact inference available to compute $\widehat{\mathbf{y}}_i$.

Another update strategy [15] is to update at each training example individually and iterate over the training data if need be. This offers no theoretical guarantees but may converge more quickly in some situations.

A third update strategy is to sum the updates until a mistake is made in the training set, then save the weights of the current model and apply the update. The number of instances that the model predicted correctly before the mistake becomes the *model weight* of the model, and predictions are made by a weighted combination of the saved models. This *voted perceptron* algorithm [28] has some nice theoretical properties, the most significant being margin-based error bound for binary classifiers.

This work is not focused on the issue of update strategy, and so I will use the first of these strategies for the remainder of this dissertation. It suffices to say that changing the update strategy for a given domain may result in better performance, though there is some empirical evidence to suggest that voting is better than simply using the final vector [28].

2.4.3 Using Kernels

One benefit of both SVM-struct and the structured perceptron is that both algorithms can use the kernel trick to free them from the constraint of learning a strictly linear model. The kernel trick [17] hinges on Mercer's theorem, which

states that there are certain functions[8] that can be expressed as an inner product in a different, possibly high dimensional space. More specifically, suppose we have two vectors, $\mathbf{a}$ and $\mathbf{b}$, and a kernel function K. Suppose further that the higher dimensional mapping corresponding to K is φ, then:

$$K(\mathbf{a}, \mathbf{b}) = \langle \varphi(\mathbf{a}), \varphi(\mathbf{b}) \rangle \tag{2.51}$$

A linearly separating boundary in the space given by φ, may be a non-linear boundary back in the original feature space. A feature of the algorithms under consideration is that they depend only on the dot product for classification. Thus, if we can replace all dot products in the algorithm with kernel evaluations, we will be learning in the kernel space rather than the original feature space, transforming the linear algorithm into one that is non-linear.

I show here that kernels can be easily incorporated into structured gradient boosting as well. First, consider that the final weight vector, $\mathbf{w}_n$ is just the initial weight vector $\mathbf{w}_0$, summed with all of the n subsequent updates, so that we have the so-called *dual form* [17] of the problem[9].

$$\mathbf{w}_n = \mathbf{w}_0 + \alpha \sum_{i=1}^{n} \frac{-\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)}{1 + \exp(\langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w}_{i-1} \rangle)} \tag{2.52}$$

where $\mathbf{w}_i$ indicates the vector that results from summing the first i terms in the summation and α is the learning rate.

[8] More specifically, the theorem refers to functions that are continuous, symmetric, and positive semi-definite.

[9] I use the uniform misclassification cost here, but the derivation generalizes easily to arbitrary misclassification cost.

At performance time and during learning, the algorithm chooses the best answer given the current model. That is, upon getting a new example $\mathbf{x}^*$ to classify, during training or in performance, it chooses the $\mathbf{y}$ that maximizes $\langle \mathbf{w}, \Phi(\mathbf{x}^*, \mathbf{y}) \rangle$. Where $\mathbf{w}$ is the current model at learning time or the final model at performance time. Equivalently, we can maximize:

$$\langle \mathbf{w}_0, \Psi(\mathbf{x}^*, \mathbf{y}) \rangle + \alpha \sum_{i=1}^{n} \frac{-\langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \Psi(\mathbf{x}^*, \mathbf{y}) \rangle}{1 + \exp(\langle \delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w}_{i-1} \rangle)} \tag{2.53}$$

Notice that the dot product is the crucial computation and at the core of the algorithm. Again supposing the existence of the kernel function K and the implied higher dimensional space φ, we first map all of the vectors into the higher dimensional space:

$$\langle \varphi(\mathbf{w}_0), \varphi(\Psi(\mathbf{x}^*, \mathbf{y}) \rangle + \alpha \sum_{i=1}^{n} \frac{-\langle \varphi(\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)), \varphi(\Psi(\mathbf{x}^*, \mathbf{y})) \rangle}{1 + \exp(\langle \varphi(\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i)), \varphi(\mathbf{w}_{i-1}) \rangle)} \tag{2.54}$$

Substituting Equation 2.51 gives us the "kernelized" form of the algorithm, and expresses the final model as a function of all of the updates.

$$K(\mathbf{w}_n, \Psi(\mathbf{x}^*, \mathbf{y})) = K(\mathbf{w}_0, \Psi(\mathbf{x}^*, \mathbf{y})) + \alpha \sum_{i=1}^{n} \frac{-K(\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \Psi(\mathbf{x}^*, \mathbf{y}))}{1 + \exp(K(\delta(\mathbf{x}_i, \mathbf{y}_i, \widehat{\mathbf{y}}_i), \mathbf{w}_{i-1}))} \tag{2.55}$$

Because Equation 2.55 is formed recursively, it looks for a moment like its computation will require an exponential number of kernel evaluations. A little thought, however, reveals that each update term can be saved at training time, and the scalar denominator of the update incorporated into the difference vector in the numerator. Thus, the final model requires only $O(u)$ kernel evaluations at performance time, where u is the number of updates.

The astute reader will note that many structured prediction applications may not admit the kernelized algorithm. In particular, if the argmax of Equation 2.11 is computed analytically, we may not be able to use kernels as these analytical algorithm tend to utilize the individual feature values and so cannot utilize the implicit mapping to kernel space. In instances where this argmax can be computed through exhaustive or heuristic search, kernels may offer important performance benefits. Such a case is described below.

2.5 Multi-class Classification: An Instructive Example

Let us consider the domain of multi-class classification as in instructive example of how structured gradient boosting can be applied. In this domain, the input space $\mathcal{X}$ is a space of fixed length vectors $\Re^d$ representing the features of the input instance. The output space $\mathcal{Y}$ is the space of the n classes that can be assigned to each input instance, $\mathcal{Y} = \mathbf{y}1, \ldots, \mathbf{y}n$. What remains is do define the misclassification cost Δ and the feature function Ψ.

The misclassification cost is reasonably straight forward: We simply use the *loss matrix* [77] of the given domain as the misclassification cost function. In the absence of a domain-specific loss matrix, we can apply $0-1$ loss[10] here and retain the generality of the approach. In this case:

$$\Delta(\mathbf{x}, \mathbf{y}_i, \mathbf{y}) = \begin{cases} 0 & \text{if } \mathbf{y}_i = \mathbf{y} \\ 1 & \text{otherwise} \end{cases} \tag{2.56}$$

Finally, we must define the Ψ function. Although there are many definitions that may be adequate, I will use the winner-take-all formulation followed in structural support vector machines [16, 92]. First, I define a binary vector representation $\Lambda^c(\mathbf{y}) \in \{0,1\}^n$ for class labels in this domain where the ith element of the binary vector is 1 if and only if $\mathbf{y} = \mathbf{y}i$, and zero otherwise. For example, if $n = 3$, $\Lambda^c(\mathbf{y}2) = 010$.

The feature vector will then be a the tensor product, $\otimes$, of this binary vector with the original feature vector. For those unfamiliar with this operation:

$$\otimes : \Re^d \times \Re^k \mapsto \Re^{dk}, (\mathbf{a} \otimes \mathbf{b})_{i+d(j-1)} = a_i b_j \tag{2.57}$$

And the feature function is:

$$\Psi(\mathbf{x}, \mathbf{y}) = \mathbf{x} \otimes \Lambda^c(\mathbf{y}) \tag{2.58}$$

[10]that is, a loss matrix with ones in every location except along the diagonal where there are zeros.

This is provably equivalent [93] to the "stack of vectors" representation for multi-class classification [16]. Essentially, if there are d features in the original input vectors and n classes, each of the n segments of length d that comprise $\mathbf{w}$ correspond to one of the classes in the output space. For a given segment, each element represents the "influence" (positive or negative and how much) that the corresponding element of the input vector contributes to the likelihood of the given class being correct.

With the functions Ψ and Δ in place, all methods of structured prediction apply directly, including those of structured gradient boosting. Note that the inference routine - computing the argmax of Equation 2.11 - is easy in this case if n is even reasonably small, which is true for most multi-class classification problems: one can simply enumerate the classes, compute the inner product for each of them, and select the best one. This step will not be so easy in future applications and will require either search or a more clever analytical computation of the argmax.

Tables 2.1 and 2.2 shows results on some of the datasets from the UCI machine learning repository [5] . In particular, I use the *ionosphere* [84], *iris* [27], *sonar* [32], *balance-scale* [83], *pima-diabetes* [87], and *waveform-5000* [10] data sets. As points of comparison, I use the *classical support vector machine* [33], *1-nearest neighbor* [2], *naïve Bayes* [41], and *decision tree* [71] algorithms. I use the implementations of these algorithms found in the WEKA [98] data mining package. Error rate is computed using 10-fold cross validation.

For gradient boosting and classical support vector machines, I use polynomial kernels. More precisely, I employ the kernel function:

	ionosphere	sonar	iris	balance-scale
Support Vector Machine	11.40	24.04	4.00	12.32
1-Nearest Neighbor	13.68	13.46	4.67	13.44
Naïve Bayes	12.38	32.21	4.00	9.60
Decision Tree	8.55	28.85	4.00	23.36
Gradient Boosting	12.42	25.96	3.73	12.01

Table 2.1: Ten-fold cross validation percent error for a selection of the UCI datasets.

	pima-diabetes	waveform-5000	average
Support Vector Machine	22.65	15.71	15.01
1-Nearest Neighbor	29.81	32.58	17.94
Naïve Bayes	23.69	16.09	16.32
Decision Tree	26.17	27.96	19.81
Gradient Boosting	34.53	16.51	17.52

Table 2.2: Ten-fold cross validation percent error for a selection of the UCI datasets and the average percent error for all six datasets in Tables 2.1 and 2.2

$$K(\mathbf{a}, \mathbf{b}) = (\langle \mathbf{a}, \mathbf{b} \rangle + 1)^3 \tag{2.59}$$

As we can see from the table, the algorithm is able to meet the performance of these standard classification algorithms in many cases. As expected, the performance of gradient boosting is most closely related to that of the classical support vector machine. Also as expected, the solutions found by gradient boosting are usually not as good as the classical SVM solutions, but in most cases the difference in error is not particularly large. The exception is the pima-diabetes dataset, which has large amounts of inconsistent data and benefits from the strict regularization of the classical SVM.

To summarize, I have developed a margin-based version of the structured perceptron update and showed that it can incorporate general notions of misclassification cost as well as kernels. I have also shown that it can be easily applied to multiclass classification. Next, we will see how this update can be put to use to solve structured machine learning problems.

Chapter 3 – Learning from Demonstrations: A Simple Application

Here I show a reasonably straightforward and successful application of structured gradient boosting[1]. I attempt to use gradient boosting to perform *learning by demonstration*. In this paradigm, a "teacher" presents a "student" with a plan to accomplish a given goal, usually formalized in machine learning literature as a sequence of actions. The student can then generalize the world state to which the demonstrated plan applies to other states where the plan may also apply.

Often, the demonstration plan is one of an exponential number of plans that may satisfy a given goal set, and in many domains (such as routing and scheduling), satisfying the goals of planning may be almost trivial. The higher achievement then, is to find a plan that satisfies the goal set optimally, or at least much better than the average, randomly drawn, goal-satisfying plan. Implicit in the above description is the notion that the demonstrated plan is one such "optimal" or "much better than average" plan.

In the reinforcement learning literature, the student typically learns through exploration. The student is allowed to take random actions in the world, and whenever one of these actions is taken, a reward is given. Over the course of many thousands of random actions, it becomes clear to the student which actions and world states generate the most reward. The best sequence to accomplish the

[1]This work was originally presented at the *AAAI '07 Workshop on Acquiring Planning Knowledge via Demonstration*.

given goal, then, becomes the sequence of actions that takes the student through the sequence of world states with the highest reward. In this setting, learning by demonstration guides exploration by "showing" the student a number of high-utility states, thus eliminating the need to discover them by random exploration.

The primary goal of learning in this setting, then, is to generalize the demonstrations given in the training set to states not seen in the demonstration. One simple way of doing this is to use the demonstrations to estimate the long-term reward obtained by taking a given action in a given world state. Clever, feature-based representations of this value function allow generalization over the state space, but this process still learns the objective function *indirectly*. That is, the above approach learns a value function over the entire state space and then attempts to maximize the value of constructed path.

Using structured gradient boosting, I will construct a discriminative approach to this problem that learns the objective, a function that discriminates good plans from bad ones, directly.

3.1 Related Work

This work is related to two threads of work in machine learning. The first is inverse reinforcement learning [63]. Here it is assumed that the demonstrated behavior is the result of optimally solving a Markov Decision Process (MDP). The task is to learn the unknown reward or cost function of the MDP from the demonstrated trajectories of its optimal solution. One approach to this problem is to assume

that all the other trajectories to be suboptimal and learn reward functions which maximally distinguish the optimal trajectories from the suboptimal ones. Since the number of suboptimal solutions is exponential in the size of relevant parameters, this problem is similar to the structured prediction task and is tackled by a similar iterative constraint generation approach. In each iteration, the MDP is solved optimally for the current reward function, and if the optimal solution generates a trajectory different from the demonstrated trajectory, it is used to train the next version of the reward function which maximally separates the optimal trajectories from the suboptimal trajectories [1, 75].

The task I study in this chapter, however, is more naturally formulated as learning to act from demonstrations [43]. Unlike inverse reinforcement learning that tries to learn the reward function, thus indirectly defining an optimal policy, here I directly seek to distinguish good state-action pairs from bad state-action pairs. Each state-action pair is described by a feature vector, and the optimal state-action pairs are assumed to maximize a weighted sum of its features. Thus, learning the weights of this optimizing function is sufficient to generate optimal behavior. Unlike in inverse reinforcement learning, the weights need not correspond to reward values. They merely need to distinguish good actions from bad actions as well as possible.

3.2 Structured Gradient Boosting for Plan Optimization

Our problem can be formulated as a structured prediction problem as follows: The "examples", $\mathbf{x}$ are *states* $s \in \mathcal{S}$ in this domain. The "class labels" $\mathbf{y}$ are *actions*, $a \in \mathcal{A}$. I will assume the misclassification cost Δ is uniform[2] in this case, for simplicity.

The approach proceeds as follows: A set of "demonstrations" that take the world from one state to another in a way that is optimal or near-optimal is given as training data. I then attempt to iteratively learn a parameterized linear function that correctly discriminates the optimal demonstration action from one drawn at random. In each iteration, the algorithm selects, from a group of random actions, the best "alternative" to each demonstration action given the current function. Based on the demonstrations and the alternatives (that are to be avoided), the algorithm computes a gradient at each parameter and take a step in this direction, ideally away from the alternatives and toward the demonstrations.

To formalize this, I first define the function $\Psi(s, a)$, which is the joint feature vector that may depend on s, a, and/or the state of the world that results from the execution of a in s. We seek a set of weights $\mathbf{w}$ that gives a higher value to the demonstration action a_i than to all other actions, given the state s_i. Specifically, suppose that $\widehat{a}_i \in A$ is the best non-optimal action given the current weights:

[2] "Uniform loss" in this case can either mean $0-1$ loss, or a constant loss of 1 regardless of the predicted class. Although the latter is semantically unusual, it has the mathematical consequence of updating even when the correct answer is predicted. 0-1 loss only updates when the predicted class is incorrect. Here we use constant loss of 1.

$$\widehat{a}_i = \underset{a \in A, a \neq a_i}{\text{argmax}} \langle \mathbf{w} \cdot \Psi(s_i, a) \rangle \tag{3.1}$$

The weights, then, must be engineered so that, for s_i,

$$\langle \mathbf{w}, \Psi(s_i, \widehat{a}_i) \rangle < \langle \mathbf{w}, \Psi(s_i, a_i) \rangle \tag{3.2}$$

for all demonstrations $\{s_i, a_i\} \in \mathcal{T}$. This is a basic restatement of the familiar structured prediction goal. Our margin m_i at each training example $\{s_i, a_i\} \in \mathcal{T}$ can be stated without much difficulty:

$$m_i = \langle \mathbf{w}, \Psi(s_i, a_i) \rangle - \langle \mathbf{w}, \Psi(s, \widehat{a}_i) \rangle \tag{3.3}$$

And the loss L at a single training example (s_i, a_i) directly follows, according to Equation :

$$L = \log(1 + \exp(\langle \mathbf{w}, \Psi(s_i, \widehat{a}_i) \rangle - \langle \mathbf{w}, \Psi(s, a_i) \rangle)) \tag{3.4}$$

Define the following notation for convenience:

$$\delta(s_i, a_i, \widehat{a}_i) = \Psi(s_i, a_i) - \Psi(s_i, \widehat{a}_i) \tag{3.5}$$

Finally, suppose our current weight vector is $\mathbf{w}_k$. The gradient for the loss expression can be derived at each feature in the representation as follows:

$$\nabla L = \frac{dL}{d\mathbf{w}} \tag{3.6}$$

$$= \frac{-\delta(s_i, a_i, \widehat{a}_i) \exp(\langle \mathbf{w}, \Psi(s_i, \widehat{a}_i) \rangle - \langle \mathbf{w}, \Psi(s, a_i) \rangle)}{1 + \exp(\langle \mathbf{w}, \Psi(s_i, \widehat{a}_i) \rangle - \langle \mathbf{w}, \Psi(s, a_i) \rangle)} \tag{3.7}$$

$$= \frac{-\delta(s_i, a_i, \widehat{a}_i)}{1 + \exp(\langle \mathbf{w}, \Psi(s_i, a_i) \rangle - \langle \mathbf{w}, \Psi(s, \widehat{a}_i) \rangle)} \tag{3.8}$$

The new cost function is then $\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha \nabla L$ where α is the step size parameter. We can then choose a new $\widehat{a}_i$ for each training example and recompute the gradient to get an iteratively better estimate of $\mathbf{w}$. Once the iterations are complete, and we have a final weight vector, $\mathbf{w}_f$, we have successfully constructed the function f from the problem formulation above:

$$f(s) = \underset{a \in A}{\operatorname{argmax}} \langle \mathbf{w}_f, \Psi(s, a) \rangle \tag{3.9}$$

3.3 Empirical Evaluation

I perform experiments in the *Wargus floor planning* domain described below. My general approach is to design several, not necessarily linear, objective functions in this domain and attempt to learn them using the method described above. These experiments show that learning a linear function in several simple features is sufficient to approximate the behavior of these more complex objectives, even where many of the features given are irrelevant.

3.3.1 The Wargus Floor Planning Domain

Wargus is a real-time strategy game simulating medieval warfare. A subproblem in Wargus is the planning of a military base whereby the layout of the buildings maximizes certain quantitative objectives. In general, the goals are to maximize the influx of resources and to survive any incoming attack. Figure 3.1 shows some examples.

More specifically, I consider a simplified version of Wargus in which there are two types of natural features on the map, which is an $n \times n$ grid. The first is a gold mine, and the second is a forested area. On each generated map, there is one randomly placed mine and four randomly placed forested areas. Our goal is to place four buildings on the map so that our objective quantities given below are optimized. These buildings are a town hall, a lumber mill, and two guard towers. The town hall is a storage building for mined gold. The lumber mill serves the same function for cut lumber. The towers are able to fire cannon in a given radius, providing defenses for the base. Demonstrations take the form of a randomly drawn map with the building optimally placed according to the objective function.

I postulate three such quantitative objectives based on user experience. For a given map and placement of buildings, I calculate a number between zero and one as a measure of how well each of these goals are satisfied.

- **Defensive Structure:** In the case where there is a clear part of the map from which an attack might originate, as much of this area as possible should be covered by the attack area of the guard towers. Formally, suppose that

$t_x(g)$ returns 1 if grid square g within the attack radius of tower x and zero otherwise. If the battle front of a given map is composed of squares $g_1, \ldots, g_m$, then the defensive quality d of a map with two towers is

$$d = \frac{\sum_{i=1}^{m} t_1(g_i) + t_2(g_i)}{2m} \tag{3.10}$$

- **Base Cohesion:** It is beneficial to locate buildings close to one another. This makes the base easier to defend from attack. Formally, if the locations of the buildings are $b_1, \ldots, b_4$ then the cohesion quality c is computed as

$$c = \frac{\sum_{i=1}^{4} \sum_{j=i+1}^{4} 2n - ||\mathbf{b}_i - \mathbf{b}_j||_1}{12n}. \tag{3.11}$$

  The factor $2n$ is the maximum distance possible between any two entities on the map.

- **Resource Gathering:** The lumber mill should be located to minimize the average distance between itself and the various forested areas, and the town hall should be located as closely as possible to the mine. Formally, suppose the town hall is at $\mathbf{t}$, the gold mine at $\mathbf{m}$, the lumber mill at $\mathbf{l}$, and the four forested areas at $\mathbf{a}_1, \ldots, \mathbf{a}_4$. The resource gathering quality r of the base is then:

$$r = \frac{\frac{\sum_{i=1}^{4} 2n - ||\mathbf{l} - \mathbf{a}_i||_1}{8n} + \frac{2n - ||\mathbf{t} - \mathbf{g}||_1}{2n}}{2} \tag{3.12}$$

The total solution quality is computed as a weighted combination of these objectives: $q = \alpha d + \beta c + \gamma r$. We can then vary α, β, and γ to obtain a variety of functions.

3.3.2 Domain Specifics

First note that in this domain, an entire plan, from start to finish, consists of a single, factored action (the placement of all buildings). Thus, this is a special case of general MDPs which allows us to unify reward function and discriminant action-value function. However, our approach directly applies to general MDPs where we can design a feature space that allows a linear discriminant function to distinguish nearly optimal and suboptimal actions in any relevant states.

Our experiments are done on a 10×10 grid. Since there are four buildings being placed, there are tens of millions of possible plans to consider for a given map. To generate a negative example for each iteration of the algorithm (the $\widehat{a}_i$ of Equation 3.2), the algorithm generates 10000 random plans and chooses the best one according to the current model. The plans are pre-screened so that they are valid placements (i.e., so that multiple buildings are not located on the same grid square).

Given this, note that it is impossible to receive a perfect quality score of one on all of these measures. For example, to achieve perfect quality on the resource gathering measure, the lumber mill would have to be located on the same grid

square as all of the forested areas, which would also have to be located on the same grid square.

The features in the model are of two types. First, there is a feature for the Manhattan distance between each building and each other entity on the map, resulting in $4 \times 9 = 36$ features. There are also features for the distance from each building on the map to the closest battle front square, which results in four more features, for a total of 40 features. Note that many of these features (the distance from either tower to any of the forested areas, for example) are irrelevant to plan quality.

3.3.3 Experimental Results

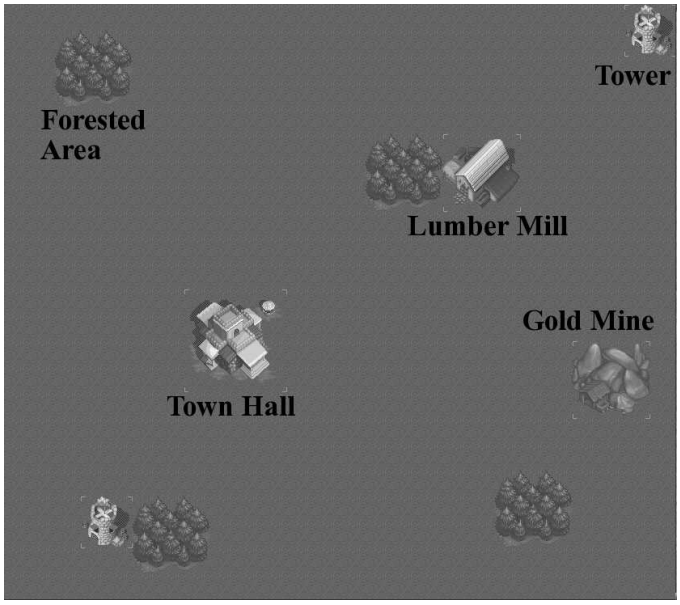
In Figure 3.2 we see the results of boosting a random model for 30 iterations according to our algorithm. We evaluate the model at each iteration on 20 different random maps by choosing for each map, according to the model, the best in a random sample of 10000 plans. The chosen plans are then evaluated according to the optimal model. As the iterations of the algorithm progress along the horizontal axis, the quality of the plan chosen by the model increases, as expected.

For reference, I plot the performance of the optimal model as well as the performance of a linear model with its weights randomly initialized, evaluated in the same way as the boosted model. Note that the score of the optimal model varies due to the fact that first, the optimal score of a map varies from map to map, and second, the optimal plan may not be in the random sample. As can be seen

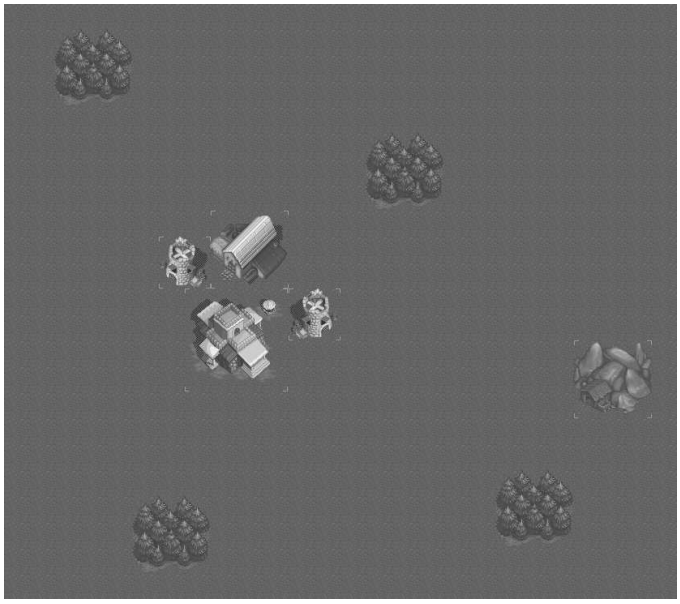
from the plots in Figure 3.2, however, much of this variability is removed as our experiments are repeated and averaged over ten trials.

In every case, the boosted function is able to learn a floor planning algorithm that is closer to optimal than random. This is true in particular for Figure 3.2(b), where the performance of the model converges to performance extremely close to the optimal. This is because the cohesion and resource gathering quality measures are almost directly expressible as linear functions of the given features. The defense measure is not readily expressible as a linear function. However, we see in Figure 3.2(d) that we are even able to learn a reasonable model when the defense measure is the only component of the objective. Finally, in 3.2(a), we see that that model performs admirably when it is forced to trade off all of the various components of the objective against one another.

Figure 3.3 shows learning curves for two of the objective functions from Figure 3.2. The number of training maps is plotted along the horizontal axis. Again, as expected, more training examples improves performance. We see that, again, we are able to learn more quickly when the defense measure is removed from the objective. More important to note, however, is the scale of the horizontal axis. For both objective functions, the algorithm is able to learn good models with only 10 to 15 training traces, even in the presence of many irrelevant features.

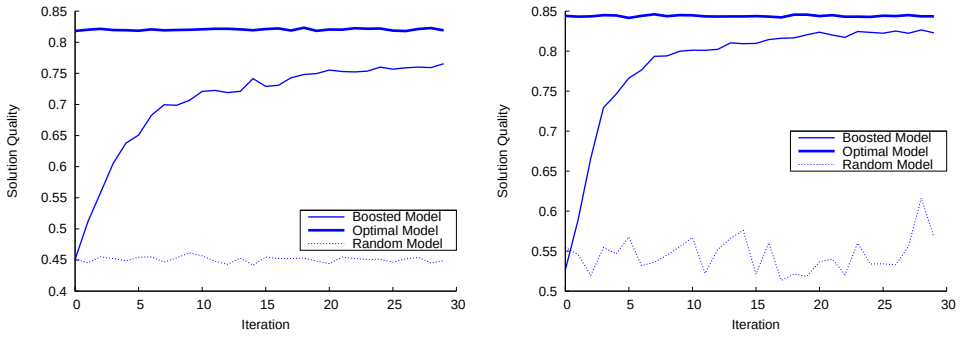


(a) An example of poor base cohesion.



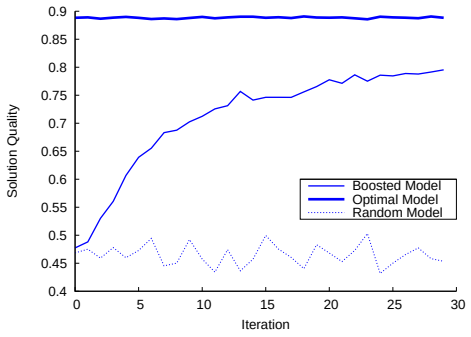
(b) An example of good base cohesion.

Figure 3.1: Examples of floor plans in the Wargus domain.

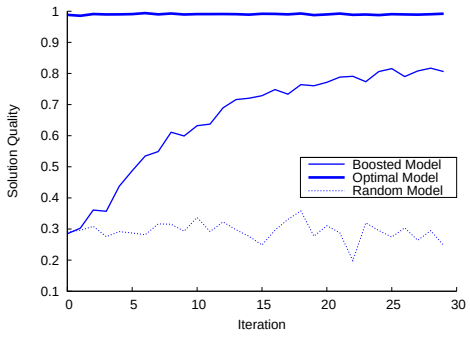


(a) $\alpha = 0.33, \beta = 0.33, \gamma = 0.33$

(b) $\alpha = 0, \beta = 0.5, \gamma = 0.5$

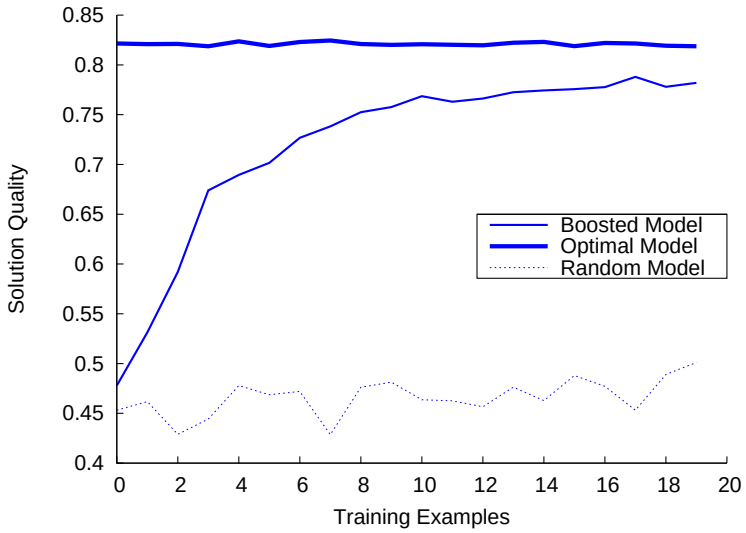


(c) $\alpha = 0.5, \beta = 0, \gamma = 0.5$

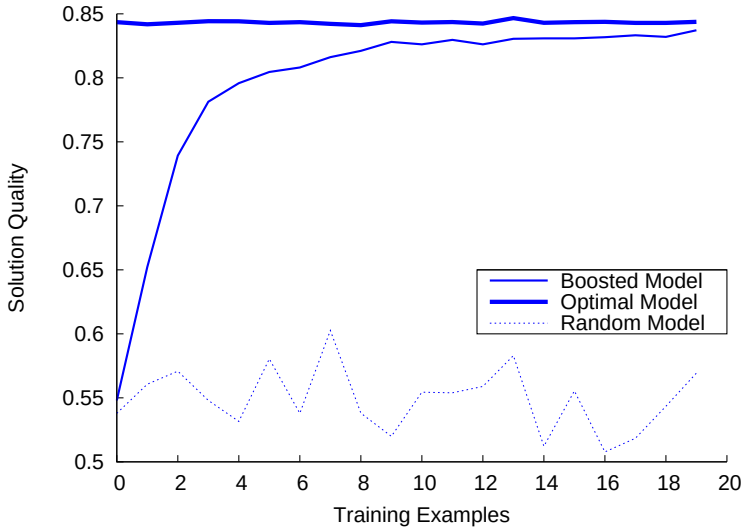


(d) $\alpha = 1, \beta = 0, \gamma = 0$

Figure 3.2: Boosting curves for four objective functions in the Wargus floor planning domain. The training set contains 15 maps.



(a) $\alpha = 0.33, \beta = 0.33, \gamma = 0.33$



(b) $\alpha = 0, \beta = 0.5, \gamma = 0.5$

Figure 3.3: Learning curves for two objective functions in the Wargus floor planning domain.

Chapter 4 – Sequence Matching and Retrieval

Sequence alignment and retrieval has grown to become a problem of considerable interest in the computer science literature. The problem, briefly stated, is as follows: We are given a target set of sequences and a query sequence that is a noisy version of one of the targets. We must design an algorithm that maps the query to the correct target for all possible query sequences.

The sequences can be any type of sequentially ordered data. In biological sequence analysis [25, 4, 3], we are given a query protein with unknown secondary structure (sequence of amino acids) and a target set of sequences with known secondary structure. By finding the "correct" match in the target set, we can predict the secondary structure of the unknown protein. In speech recognition [72] our target set is a dictionary of phoneme sequences, representing words, and the query is a phoneme sequence spoken by a person. The paradigm has also been applied to plant identification [85] and music information retrieval [82].

Here, I present an introduction to sequence matching and the adaptation of two previous approaches to solving this problem.

4.1 Dynamic Programming Sequence Alignment

Our intuition for sequence alignment is shown in Figure 4.1, using the words `CABIN` and `DRAIN`[1]. We wish to line up letters that match, and consider the ones that only appear in one sequence or the other as lining up with nothing at all. Let us introduce some terminology to describe the situation. Suppose we call the word `CABIN` the target sequence and the word `DRAIN` the query. If, in the given alignment, there is a target character that pairs with no query character we will call this a *deletion*, as it is not found in the query. Similarly, if there is a query character that pairs with no target character, we will call this an *insertion*. If two characters are corresponding, they are called a *match*. Thus, each of the seven steps in the alignment below can be marked with an **i**, **d**, or **m** as is appropriate.

i	**i**	**d**	**m**	**d**	**m**	**m**
-	-	C	A	B	I	N
D	R	-	A	-	I	N

Figure 4.1: A Lexicographical Example of Sequence Alignment

The original dynamic programming algorithm for sequence alignment is attributed variously to Smith and Waterman [88] and Levenshtein [48] and is based on finding highest probability alignment of the two sequences. What exactly I mean by highest probability will be defined momentarily.

Suppose we have two sequences, **t** and **q**. Let us further suppose that we know a distribution P that describes the relative frequencies of certain events occurring

[1]With thanks to [57] for the example.

in the alignment of these two sequences. For example, $P(t, q)$, is the probability that character t will match character q in a given sequence alignment, $P(-, q)$, the probability that the character q in the query was inserted and does not correspond to anything in the target, and $P(t, -)$ is the probability that the character t was deleted in the target and corresponds to nothing in the query.

Now we can consider the basic recursion. If we start at the beginning of both sequences, by considering t_1 and q_1, we see that we only have three options for this pair of characters: If the two characters match, we move on to the next pair of characters. If q_1 was inserted, we next consider t_1 and q_2, and if t_1 was deleted, then we next consider t_2 and q_1.

Since we do not know which of these three will be the best choice in the long run, we simply compute all three alternatives and choose the best one. Now we can construct the most probable alignment of $\mathbf{t}$ and $\mathbf{q}$, starting at t_i and q_j:

$$\text{maxAlign}(i, j, \mathbf{t}, \mathbf{q}) = \max \begin{cases} P(t_i, q_j) * \text{maxAlign}(i+1, j+1, \mathbf{t}, \mathbf{q}) \\ P(-, q_j) * \text{maxAlign}(i, j+1, \mathbf{t}, \mathbf{q}) \\ P(t_i, -) * \text{maxAlign}(i+1, j, \mathbf{t}, \mathbf{q}) \end{cases} \tag{4.1}$$

with the base cases:

$$
\begin{aligned}
i = |\mathbf{t}| \wedge j = |\mathbf{q}| &\rightarrow \text{maxAlign}(i, j, \mathbf{t}, \mathbf{q}) = 1 \\
i = |\mathbf{t}| \wedge j \neq |\mathbf{q}| &\rightarrow \text{maxAlign}(i, j, \mathbf{t}, \mathbf{q}) = P(-, q_j) * \text{maxAlign}(i, j+1, \mathbf{t}, \mathbf{q}) \\
i \neq |\mathbf{t}| \wedge j = |\mathbf{q}| &\rightarrow \text{maxAlign}(i, j, \mathbf{t}, \mathbf{q}) = P(t_i, -) * \text{maxAlign}(i+1, j, \mathbf{t}, \mathbf{q})
\end{aligned}
\tag{4.2}
$$

and the distance function becomes the negative logarithm of the highest probability alignment of the two sequences, starting from the beginning of each one:

$$
d(\mathbf{t}, \mathbf{q}) = -\log(\text{maxAlign}(1, 1, \mathbf{t}, \mathbf{q})) \tag{4.3}
$$

Of course, this recursion is exponential in the lengths of both $\mathbf{t}$ and $\mathbf{q}$. Equally obvious is the dynamic program that makes the algorithm $O(|\mathbf{t}||\mathbf{q}|)$: We simply fill a $|\mathbf{t}| \times |\mathbf{q}|$ matrix with the score of the best possible alignment up to that point as it is computed. Future recursive calls first reference the matrix rather than doing redundant computation.

In some domains, we would like to compute alignments that do not penalize prefixes or suffixes in the target or query as is appropriate, and this is easily achieved by modifying the base cases for the recursion above. For example, if we add the condition that $\text{maxAlign}(i = |\mathbf{t}|, j, \mathbf{t}, \mathbf{q}) = 1$ we have a version of the function that does not penalize suffixes in the query. When i reaches the end of the target, j will continue to the end of the query with no penalty. This is gone over in more detail in [57].

Some interesting extensions to this basic algorithm are discussed in [61] where it is posited that there should be another branch in the above recursion for the case where a single event in the target maps to two events in the query:

$$P(t_i, q_j, q_{j+1}) * \text{maxAlign}(i+1, j+2, \mathbf{t}, \mathbf{q}) \tag{4.4}$$

This condition is referred to as *fragmentation*. Similarly, *consolidation* refers to the condition where a single event in the query matches two events in the target:

$$P(t_i, t_{i+1}, q_j) * \text{maxAlign}(i+2, j+1, \mathbf{t}, \mathbf{q}) \tag{4.5}$$

This can be taken to its logical extension, whereby we consider matching $\{0, \ldots, n\}$ in the target to $\{0, \ldots, n\}$ events in the query, giving $(n+1)^2$ branches in the recursion. This idea is gone over in detail in [97]. The evidence presented in these papers is domain-specific and only marginally convincing. For the remainder of this dissertation, I will use only the three types of events described above (insert, delete, and match).

Another interesting point is made by [25]: Often *ad-hoc* penalty methods are used in the context of this algorithm. For example, one may define a set of *costs*, c, as a substitute for the underlying distributions:

$$c(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x = \text{- or } y = \text{-} \\ 2 & \text{otherwise} \end{cases} \tag{4.6}$$

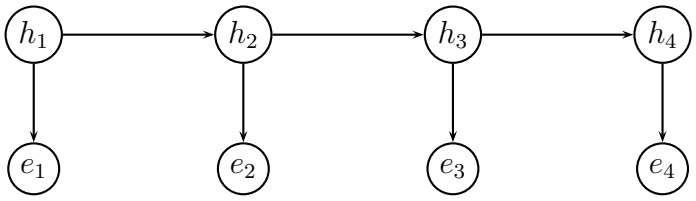


Figure 4.2: The basic hidden Markov model

with the Smith-Waterman algorithm redefined such that first, we add rather than multiply the values from the recursion, and second we choose the *lowest cost* alignment rather than the *most probable* alignment.

4.2 Hidden Markov Models

One of the chief techniques that has overtaken the traditional dynamic programming method of sequential alignment is the Hidden Markov Model. Here I give a basic outline, again assuming the reader is at least somewhat familiar with Bayesian inference. For those who are looking for a survey with more depth, they may consult [78] for a general overview or [72] for a more sequence-specific discussion.

The standard Hidden Markov Model is a sequence of *hidden states* and *emission* or *observation states*, as shown in Figure 4.2. The states in the model correspond to characters in the sequence. The hidden states correspond to a target sequence and the emission states to a query.

There are a few obvious difficulties here: First, the prototypical HMM assumes that the lengths of the target and query match - that there are no insertions or deletions. Second, note that this model requires two separate distributions,

$P(e_i, h_i)$ just as defined above, but also $P_t(h_i, h_{i+1})$, the *transition model*. This gives us yet another set of parameters in this model that must be either learned or estimated by hand. The former difficulty will be dealt with momentarily.

To find the probability of a sequence of emissions, $\mathbf{e}_{1:t} = \{e_1, \ldots, e_t\}$, we simply compute the probability of the initial hidden state, $P(\mathbf{h}_1, e_1)$, then pass the so-called "forward message" on to the next state: First we update $P(\mathbf{h}_2)$ given our new values for $P(\mathbf{h}_1)$:

$$P(\mathbf{h}_2, e_1) = \sum_{\mathbf{h}_1} P(\mathbf{h}_2|h_1)P(h_1, e_1) \tag{4.7}$$

where the use of non-boldface type indicates that each possible value of that variable is considered.

The second step is to estimate the joint distribution of $\mathbf{h}_2, e_1, e_2$ using the new evidence:

$$P(\mathbf{h}_2, e_1, e_2) = P(\mathbf{h}_2, e_1)P(e_2|\mathbf{h}_2) \tag{4.8}$$

Obviously, we can continue this process recursively, for t steps, until we obtain $P(\mathbf{h}_t, \mathbf{e}_{1:t})$. We then simply sum over $\mathbf{h}_t$ to obtain the likelihood of the emissions (query) with respect to the hidden states (target).

The shortcoming here, of course, is the lack of support for inserted and deleted characters. There are several model topologies that resolve this difficulty, but one of the more common ones is shown in Figure 4.3. We have simply added insertion

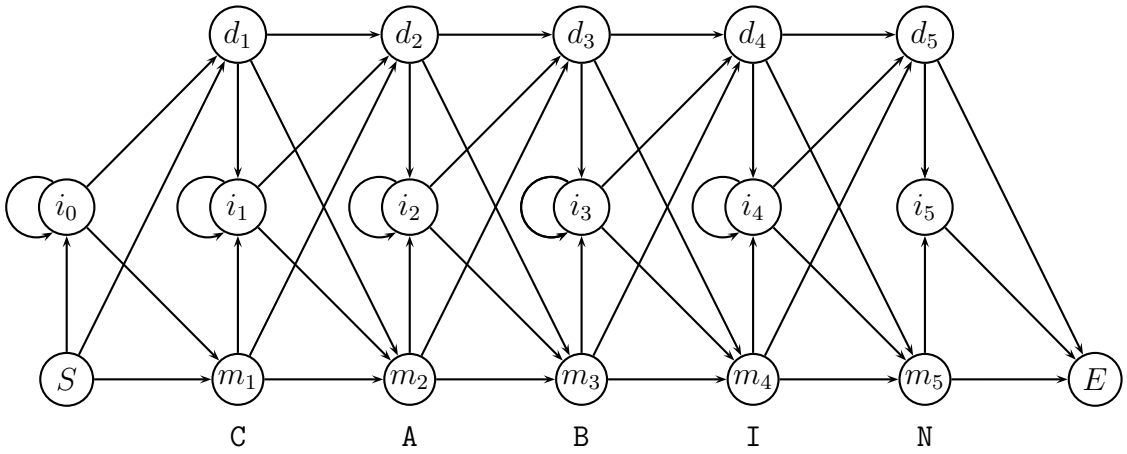


Figure 4.3: A Model Topology for Matching Arbitrary-length sequences to the word `CABIN`.

and deletion states at each character in the target model[2]. The distributions over these states are populated by the distributions $P(-, q)$ and $P(t, -)$.

This type of model describes the process of generating an arbitrary length query sequence from a target sequence. In this particular case, the model is describing the process of generating a query sequence given the target sequence `CABIN`. Figure 4.3 is the model of a particular target in our database. Suppose we use this model to explain the process of generating the word `DRAIN` given this target model. In Figure 4.1, we have the alignment of the two words. Recall that each pair of characters in the alignment can be described as a match, insertion, or deletion. We thus have a sequence of *alignment events*, in this case {**i i d m d d m m**}. If we have constructed our model correctly, this sequence will specify a unique path

[2]Note that this model is for a target sequence of length 5, but can be extended to any length without loss of generality.

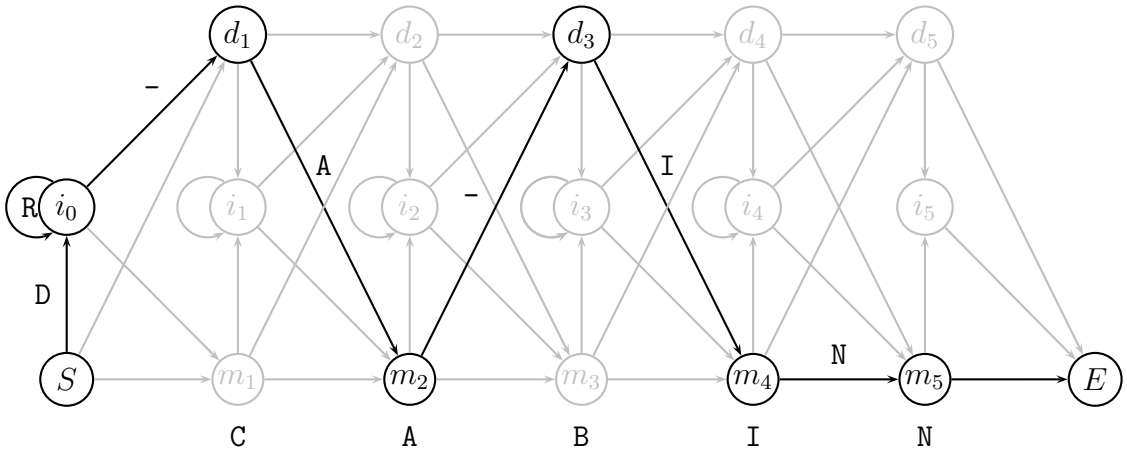


Figure 4.4: The Generation of the Query DRAIN from the Target Model for CABIN.

through the model. We show this path in Figure 4.4 with the emitted characters on the transition arcs.

It is straightforward to explain the generation of the query from target model. First, before we encountered the first letter in the target, we inserted D and R. Then, we deleted the first character, C, from the target. Then, the letter A in the target generated a matching A in the query. We can proceed like this until we reach the end of the path.

Note that the mathematics has not changed much. The emission states that should be connected to the m and i-type states[3] of the target model have been combined with the hidden states in the interest of simplicity, but we still compute probabilities of query given target. That is, m states and i states consider the current query character as well as the current target character. This gives us a

[3] d-type states, of course, do not emit a character - remember that delete states correspond to target characters that are not paired with query characters.

little extra bookkeeping to do. For the traditional model in Figure 4.2, we simply step forward, update the probabilities at the current state, and step again. Here, the step at each state goes three ways, and we may or may not "burn" a query character at each step. This sounds like it becomes exponential - but often as we step forward, paths will meet, and we can simply sum their probabilities together at that point. In fact, since there are $3|\mathbf{t}|+3$ states in any target model, and only $|\mathbf{q}|$ places in the query, we retain $O(|\mathbf{t}||\mathbf{q}|)$ complexity for the forward algorithm.

So we know that we can use the forward algorithm on this model by keeping track of where we are in the query, making sure to go in all possible directions, and summing redundant paths as we go. However, let us be a little more precise about the mathematics involved here, and relate it to the Smith-Waterman algorithm.

4.3 Discussion

Although I have explored two different models, they are actually quite similar. First, let us assume that, rather than summing the likelihood over all possible paths (or alignments) through the hidden Markov model, we pick the most probable one as an approximation. This is the *Viterbi algorithm* and requires only a modest extension to the forward algorithm described above[4]. This gives us, as in the top row of Figure 4.1, a list of alignment states $\mathbf{a}$ through the target model $\mathbf{t}$, and a list of symbols in the query sequence, $\mathbf{q}$. Define $q_{l(i)}$ as the query character

[4]The extension is, in essence, that we must remember the most likely path up to the current timestep as we advance through the model. See [78] for more details.

corresponding to the ith state in the alignment and $t_{m(i)}$ as the corresponding target character. For example, $q_{l(2)} = \texttt{R}$ and $t_{m(6)} = \texttt{I}$.

The probability of the list of states with respect to the query, then, is just a product over the probability of each state:

$$P(\mathbf{q}, \mathbf{a}|\mathbf{t}) = P(E|a_{|\mathbf{a}|}) \prod_{i}^{|\mathbf{a}|} P(a_i|a_{i-1}) P(q_{l(i)}|a_i, t_{m(i)}) \tag{4.9}$$

where $P(E|a_{|\mathbf{a}|})$ is the probability of the final transition into the *end state* of the model shown in Figure 4.3.

Note that if a_i is a delete state, there is no observation $P(q_{l(i)}|a_i)$ and we set this term to $P(t_{m(i)}, -)$ If a_i is an insert state, then $P(q_{l(i)}|a_i) = P(-, q_{l(i)})$ and if a_i is a match state, $P(q_{l(i)}|a_i) = P(q_{l(i)}|t_{m(i)})$. To be a little more formal:

$$P(q_{l(i)}|a_i, t_{l(i)}) = \begin{cases} P(t_{l(i)}, -) & \text{if } a_i \text{ is a delete state} \\ P(-, q_{l(i)}) & \text{if } a_i \text{ is a insert state} \\ P_m(q_{l(i)}|t_{m(i)}) & \text{if } a_i \text{ is a match state} \end{cases} \tag{4.10}$$

The transition probabilities are split similarly. More details can be found many other places in the literature [46].

Moving to the Smith-Waterman algorithm, the basic recursion tells us that the maximal path through the model will also be a simple product over the list of "states" in the alignment. Again, if the algorithm returns an alignment $\mathbf{a}$ of a target $\mathbf{t}$ and a query $\mathbf{q}$:

$$P(\mathbf{q}, \mathbf{a}|\mathbf{t}) = \prod_{i}^{|\mathbf{a}|} P(a_i) P(q_{l(i)}|a_i, t_{m(i)}) \tag{4.11}$$

where we define $P(q_{l(i)}|a_i, t_{m(i)})$ just as above. If we assume that all alignment events are equally likely given no evidence, then $P(a_i)$ drops out and we are left with the recursion in Equation 4.1.

Thus we see that, in the end, both of these techniques require the same thing: a set of probabilities or costs that give properly discriminating distances between target and query sequences. Throughout these last sections, I have assumed that we have access to the "true" generative distribution of the domain, but this is almost never the case. Usually, these parameters must either be estimated by hand [59] or learned from data. As the former requires significant engineering by a domain expert, the latter becomes the preferable method.

4.4 Learning Distance Functions

In the previous chapter, we saw that whether sequences are aligned using the Smith-Waterman algorithm or aligned using hidden Markov models, the correct specification of the parameters of these algorithms is crucial to their success. Ideally, we would like to specify parameters that promote proper discrimination between correct targets and incorrect ones. That is, we would like the distances computed by our alignment algorithms to be low when the distance is between a query sequence and the correctly matching target, and high otherwise.

Unfortunately, the specification of these parameters may require expertise in the domain and significant time. A better approach, it seems, is to learn the correct parameters (costs) for these models from training data. Below, I discuss two methods for learning these costs, and point out some of their shortcomings.

4.4.1 Generative Learning

The generative approach to learning cost functions assumes a probabilistic model for generating queries from target sequences [25]. The model specifies the probability of various edit operations and a query is "generated" from a target sequence by applying the various operations to the target according to their probabilities. The goal of the generative learning process, then, is to produce estimates of these probabilities based on the given training alignments. Under this model, the probability of a particular alignment of two sequences is simply the product of the probabilities of each edit operation in the alignment. The distance between the sequences is then the negative logarithm of the probability of the alignment.

Under this model our cost functions become the negative logarithms of the edit operation probabilities:

$$\begin{aligned} c(t,q) &= -\log(P(q|t,e=\text{match})P(e=\text{match})) \\ c(-,q) &= -\log(P(q|e=\text{insert})P(e=\text{insert})) \\ c(t,-) &= -\log(P(t|e=\text{delete})P(e=\text{delete})) \end{aligned}$$

where $P(e)$ is the probability of generating edit operation e, $P(q|t, e = \text{match})$ is the probability that query symbol q is generated when a match operation is applied to target symbol t, $P(q|e = \text{insert})$ is the probability that query symbol q is inserted by an insert operation, and $P(t|e = \text{delete})$ is the probability that target symbol t is deleted by a delete operation. Once these costs are in place, we can use them in the "addition" version of the Smith-Waterman algorithm as described above.

Now suppose we are given a training set $\mathcal{S}$ of alignments, where each training example in $\mathcal{S}$ is a triple $(\mathbf{t}, \mathbf{q}, \mathbf{a})$ of a target $\mathbf{t}$, query $\mathbf{q}$, and corresponding alignment $\mathbf{a}$ that we interpret as the optimal alignment of $\mathbf{t}$ and $\mathbf{q}$. We can use counts of the edit events in $\mathcal{S}$ to estimate the above probabilities. For example, let $\#_{\mathcal{S}}(p)$ be the number of events in $\mathcal{S}$ for which p is true. Then to estimate c for a particular pair of characters t_i and q_j:

$$\begin{aligned} P(q = q_j | t = t_i, e = \text{match}) &= \frac{\#_{\mathcal{S}}(q=q_j, t=t_i, e=\text{match})}{\#_{\mathcal{S}}(t=t_i, e=\text{match})} \\ P(e = \text{match}) &= \frac{\#_{\mathcal{S}}(e=\text{match})}{n} \end{aligned}$$

and similarly for the cases where t_i or q_j are equal to the null character, "-". This is standard maximum-likelihood estimation for discrete probability models. Note that this formulation assumes that the query and target sequence elements are from discrete finite domains. This assumption has been predominant in generative approaches for sequence alignment.

4.4.2 SVM-align

Without too much difficulty, we can fit the sequence alignment problem into the SVM-struct formalism. The Smith-Waterman algorithm is used to compute the feature vector Ψ, whereby a target and query are optimally aligned and the events from that alignment counted into the feature vector. The parameters of the model, $\mathbf{w}$ are the costs of each possible replacement (given by $c(a, b)$ above, so that the size of $\mathbf{w}$ is roughly the square of the alphabet size[5]).

To relate this to standard edit distance learning, suppose we treat the usual inner product as an inverse distance or *similarity* metric, so that higher values of the inner product represent *decreased proximity*

$$d(\mathbf{t}, \mathbf{q}) = -\langle w, \Psi(\mathbf{t}, \mathbf{q})\rangle\rangle \tag{4.12}$$

Our goal, as expected, is to learn weights $\mathbf{w}$ such that for all query-target pairs, the distance of the correct target $\mathbf{t}_i$ is greater than that of all other targets in $\mathcal{T}$. For this purpose, SVM-align tries to find weights such that for each training instance $(\mathbf{t}_i, \mathbf{q}_i)$ we have,

$$\forall \mathbf{t} \neq \mathbf{t}_i \in \mathcal{T}, \langle \mathbf{w}, \Psi(\mathbf{q}_i, \mathbf{t}_i)\rangle - \langle \mathbf{w}, \Psi(\mathbf{q}_i, \mathbf{t})\rangle > 0 \tag{4.13}$$

This can, as shown in Section 2.3.1, be formulated as a constrained optimization problem and be solved as a series of quadratic programs, with results that have strong theoretical guarantees.

[5]It is slightly greater than this due to insertion and deletion cost.

4.4.3 Comparison of Learning Methods

I have explored in detail the history of sequence alignment and two basic methods of learning distance functions in sequence alignment domains. I will now note some of the problems with these methods and motivate a new method for learning distance functions in sequence alignment domains.

4.4.3.1 The Generative Method

There are at least two problems with the generative learning method: First, the generative training of the model does not take advantage of the discriminatory nature of the underlying task, and only correct target-query alignments are used in the training process. For example, if there is a small set of features that can uniquely identify a target, it makes sense to learn to recognize them, rather then having to learn a full generative model, which might be quite complicated. There are many examples from the machine-learning literature showing that discriminative training often significantly outperforms generative approaches [47, 91].

Second, it seems that there is no way to extend the learning process so that the reward behavior of one sequence element can be applied to another, except in domain-specific instances. That is, the sequence elements are treated as atomic entities with no internal structure that facilitates generalization. This generalization crucial in domains where the number of characters in the sequence alphabet is high (or infinite) and there is little training data available. In these cases previous

approaches have resorted to guessing parameter values [58] or making simplifying assumptions [66] or both [69].

I note that while most prior generative approaches have assumed that sequence elements are unstructured, it is possible to extend them to exploit structured elements. This requires selecting a suitable probabilistic model for the data types of the sequence elements. For example, if the elements are real-valued vectors, Gaussian models may be appropriate. However, selecting an appropriate model for a particular application is often quite difficult, requiring experimentation and insight into the problem. Our approach described below can be viewed as a way to avoid the need to explicitly make such choices.

4.4.3.2 SVM-align

While SVM-align provides a powerful discriminative approach to sequence alignment learning, it still does not address the second concern from the previous subsection. Since the current implementation of SVM-align is based on the count-based feature representation described above, we still must utilize a discrete, unstructured domain for the sequence elements. Thus, we are still unable to generalize to characters not present in the training data, and we are still forced to "flatten" our representation of the sequence to a one-dimensional character set. In addition, the solving of multiple constrained optimizations requires a great deal more time than the generative approach. Although the running time is polynomial in the size

of the training set, the SVM-align formalism is still typically orders of magnitude slower than the generative approach.

Though an implementation is not available, this theoretical framework does allow for extension to more structured sequence elements. In particular, the reward functions can in general be any linear combination of features of the sequence elements. These features, however, must be hand-engineered. Such feature engineering can be quite tedious and requires significant insight into the domain. Our approach described below will attempt to address this shortcoming.

Finally, neither of these approaches attempt to address efficiency in their learning process: Although both approaches may yield cost functions that properly discriminate between correct and incorrect targets, we must still compute the distance for each target in turn in order to find the closest one. In Chapter 6, I will present a method of learning these distance functions that can optimize retrieval efficiency as well as retrieval accuracy.

4.5 Structured Gradient Boosting for Sequence Retrieval

In my motivating application of music retrieval, the sequence elements are vectors of real-valued features extracted via standard music processing. As previously stated, a straightforward application of SVM-align or standard maximum likelihood estimation requires flattening the vector space into single character elements, which as discussed previously can be undesirable. In addition, extending the above

approaches to directly utilize the element structure requires significant insight into the domain.

Here, I will apply structured gradient boosting directly to solve this problem. In particular, the vector $\mathbf{w}$ will be the replacement costs given by $c(a, b)$ above. However, by representing the gradient step ∇L as a function of which we know certain points, rather than as a vector, we will see that we are able to leverage structure and regularity within the symbol alphabet to speed up the pace of learning the distance function.

The first step in applying structured gradient boosting to this problem is to select a suitable notion of margin. A reasonable definition for the margin in this application is the difference, for a given query $\mathbf{q}_i$, between the distances to the correct target and the closest incorrect target. Suppose that $\mathbf{t}_i$ is the correct target for query sequence $\mathbf{q}_i$ and $\hat{\mathbf{t}}_i$ is the closest incorrect target for this query. If the distance between a target $\mathbf{t}$ and a query $\mathbf{q}$ is $d(\mathbf{t}, \mathbf{q})$, then the margin for query $\mathbf{q}_i$ is:

$$m_i = d(\widehat{\mathbf{t}_i}, \mathbf{q}_i) - d(\mathbf{t}_i, \mathbf{q}_i) \tag{4.14}$$

and accordingly the loss function for the query example $\mathbf{q}_i$ to be:

$$\log(1 + \exp(d(\mathbf{t}_i, \mathbf{q}_i) - d(\widehat{\mathbf{t}_i}, \mathbf{q}_i))) \tag{4.15}$$

We can express the distance function, d as a sum of the costs for the various events in the alignment. To do this, let us define define $f(a, b, \mathbf{x}_i, \mathbf{y}_i)$ to be the

number of times that character a is replaced by b in the best alignment of sequence $\mathbf{x}_i$ with sequence $\mathbf{y}_i$. Note that, in structured prediction, *higher* values for positively valued features imply that the match between input and output is *better*. In distance metric learning, the opposite is the case. To resolve this, we must negate the sum to make a semantically appropriate distance. Thus the distance function can be written as:

$$d(\mathbf{t}, \mathbf{q}) = -\sum_a \sum_b c(a, b)[f(a, b, \mathbf{t}, \mathbf{q})] \tag{4.16}$$

In the case where the alphabet is finite, we can create the feature vector $\Psi(\mathbf{q}, \mathbf{t})$ that would extract a vector of counts of replacement events, one for each replacement (a, b). Creating this vector requires the alphabet to be finite as it relies on the ability to enumerate all possible characters. In this case, there can be a one-to-one correspondence between elements of a parameter vector $\mathbf{w}$ and the function $c(a, b)$, and:

$$\sum_a \sum_b c(a, b)[f(a, b, \mathbf{t}, \mathbf{q})] = \langle \mathbf{w}, \Psi(\mathbf{q}, \mathbf{t}) \rangle \tag{4.17}$$

However, this creation of a fixed-length parameter vector is exactly the step that ties us to a discrete character alphabet. For now, I will eschew the vectors Ψ and $\mathbf{w}$ in favor of the functions f and c, respectively. The summation above can be computed efficiently even for continuous character alphabets because there will always be a finite number of events in the alignment of $\mathbf{t}$ and $\mathbf{q}$, and we may, equivalently, sum over these events rather than all possible character pairs (a, b).

With this formulation, the learning process, which is a direct application of structured gradient boosting, is to iteratively learn the function $c(a, b)$ in order to minimize the loss function. More specifically let c_k be the cost function after k iterations. Initially c_0 is set to be a human provided function, perhaps based on prior knowledge or uninformative. Given c_k the algorithm computes c_{k+1} by approximating the functional gradient ∇L_{k+1} of the loss function with respect to c_k and then setting $c_{k+1} = c_k - \nabla L_{k+1}$. This tends to move c_{k+1} in a direction that decreases the loss function. The iteration repeats until a stopping condition is reached (e.g. a specified number of iterations). The key step of this process is to compute the approximate functional gradients, which I now describe.

To approximate the functional gradient δ_{k+1} the algorithm will calculate the value of this gradient at each query-target symbol pair (a, b) in the training data, noting that for large structured alphabets many possible (a, b) pairs will not appear in the training set. This provides a training set $\{\langle (a, b), \nabla L_{k+1}(a, b)\rangle\}$ which can be passed to a function approximator yielding an approximation T_{k+1} that generalizes across all possible pairs (a, b). This is essentially the approach taken in [22] for training conditional random fields.

In each iteration, then, $c_k(a, b) = c_0(a, b) - \alpha \sum_{i=0}^{k} T_i(a, b)$ so that calls to k function approximators are required for the evaluation. I give psudeocode for the learning algorithm in Figure 4.5. The key step in creating the training set is to compute the functional gradient of the loss function for a given pair.

For convenience define $\delta f_i(a, b)$ for training example i to be,

Figure 4.5: The accuracy boosting algorithm

$\mathcal{T}$ is a set of target structures, $\mathcal{Q}$ is a training set of query sequences (for which correct targets are known), and c_k is the set of costs at iteration k. ∇L is set of training examples for learning the functional gradient, and COMPUTE-ACCURACY-GRADIENT is a function that returns a list of examples to be added to this set. n is the number of boosting iterations.

function BOOST-ACCURACY($\mathcal{Q}$, $\mathcal{T}$, c_k)
  for $k = 0$ to n **do**
    $\nabla L_{k+1}(a, b) = 0, \forall (a, b)$
    for all $\mathbf{q}_i \in \mathcal{Q}$ **do**
      $\nabla L_{k+1} \leftarrow \nabla L_{k+1}$ + COMPUTE-ACCURACY-GRADIENT($\mathbf{q}_i, \mathcal{T}, c_k$)
    end for
    $\mathcal{S} = \{((a, b), \nabla L_{k+1}(a, b))\}, \forall a, b$
    $T_{k+1} \leftarrow$ LEARN-REGRESSION-TREE($\mathcal{S}$)
  end for
end function

$$\delta f_i(a, b) = f(a, b, \mathbf{t}_i, \mathbf{q}_i) - f(a, b, \widehat{\mathbf{t}}_i, \mathbf{q}_i,) \tag{4.18}$$

With these definitions, we can combine Equations 4.15 and 4.16 and derive the functional gradient at pair (a, b) is derived as follows:

$$\nabla L_(a, b) = \frac{\partial L}{\partial c_k(a, b)} \tag{4.19}$$

$$= \frac{-\delta f_i(a, b) \exp(d(\mathbf{t}_i, \mathbf{q}_i) - d(\widehat{\mathbf{t}}_i, \mathbf{q}_i))}{1 + \exp(d(\mathbf{t}_i, \mathbf{q}_i) - d(\widehat{\mathbf{t}}_i, \mathbf{q}_i))} \tag{4.20}$$

$$= \frac{-\delta f_i(a, b)}{1 + \exp(d(\widehat{\mathbf{t}}_i, \mathbf{q}_i) - d(\mathbf{t}_i, \mathbf{q}_i))} \tag{4.21}$$

Intuitively, the gradient function modifies the weights so that the training loss decreases at each iteration. In this way, I have maintained the discriminative learning aspect of SVM-align. However, my approach is much more computationally efficient and able to generalize over the character space. I accomplish both of these goals by training regression trees to approximate each functional gradient ∇L_k. Of course, any other function approximator could be applied to the training data. This bodes well given the amount of work on function approximation that can be found in the literature.

Note also the crucial representational advantage over previous approaches. Because the gradient is a function, it may take arguments that are real-valued, vector-valued, or even structure-valued. We are no longer tied to a fixed set of replacement events. The gradient trees learned may be saved until performance time, much like in the standard dual problem. They can then be treated as variably-sized weight vectors, where the size of the weight vector is dictate by the ensemble of gradient trees learned. This, in some sense, is a form of feature induction in that the space of characters is not dictated by ad-hoc assumptions, but actually arises as a side effect of the learning process.

During inference, the "dot products" between f and c_k can be efficiently computed because we may treat f as being zero for all pairs (a, b) not in the alignment of the target and query. Thus, we simply enumerate the pairs in the alignment, and sum the calls to the k function approximators for each pair to obtain the alignment score.

4.6 Empirical Analysis

This approach will first be evaluated on the synthetic domain explained below. For these tests, we use sequences of length 5 and a target set of size 50. Error is measured as the fraction of queries classified incorrectly.

4.6.1 Synthetic Domain

In [40] a synthetic sequence matching domain is introduced. In this domain, sequences are drawn from an alphabet of 20 characters. Target sequences are randomly generated over this alphabet. Query sequences are generated for a given target according to the following procedure:

1. With probability 0.2, generate a random element in the query (an *insert* event).

2. With probability 0.4, generate a *match* event where if the target element is t, the matching element in the query is $(t + 1) \mod 20$. Move to the next element in the target.

3. With probability 0.2, generate a *match* event where if the target element is t, the matching element in the query is also t. Move to the next element in the target.

4. With probability 0.2, move to the next tuple in the target (a *delete* event).

This simple domain is sufficient to test a general sequence retrieval algorithm, but we wish to test, in addition, the algorithm's ability to exploit structure in the sequence alphabet. To this end, I introduce a slightly different version of the domain above. In this new version, the elements of the synthetic target sequences are tuples (t_1, t_2) where t_1 is an integer in the range $(0, 9)$ and t_2 is an integer in the range $(0, 29)$. We generate 10 random sequences of five elements each as our target set in the accuracy experiments, and a target set of 2000 ten-element sequences for the efficiency experiments. To generate a query, we select a random target from the set. Beginning at the first tuple in the sequence, we use the following model to generate a series of query tuples of the form (q_1, q_2) and drawn from the same domain:

1. With probability 0.3, generate a random tuple in the query where $q_2 \geq 15$ (an *insert* event).

2. Else, if $t_2 < 15$, generate a *match* event. If the target tuple is (t_1, t_2), the matching tuple in the query is $(t_1, t_2 + 1 \mod 30)$. Move to the next tuple in the target.

3. Else, move to the next tuple in the target (a *delete* event).

In short, I have introduced a threshold of $t_2 < 15$ into the domain. In theory, the proposed learning algorithm will learn quickly that sequence elements with $t_2 < 15$ are likely to be matched in the target sequence. Elements with $t_2 \geq 15$ are usually insert events when present in the query, or delete events when present in

the target. This ability to generalize the cost behavior of one character to that of another will result in superior distance functions with less training data.

4.6.2 Experimental Results

Results for accuracy learning in the synthetic domain are shown in Figure 4.6. Here we see the full power of the gradient boosting approach. SVM-align is generally able to outperform the generative approach, but at a training set size of about 15 sequences, the gradient boosting method is able to learn a trick of the domain that allows it to learn a much more accurate cost function

Finally, we have the respective running times of gradient boosting and SVM-align shown in Figure 4.7. Although both appear to be linear in the size of the training set, the constants involved for SVM-align appear to be much larger, growing to nearly a half hour of training time on a training set as small as 40 sequences. Also note that SVM-align is implemented as highly optimized C, whereas the tests for gradient boosting were run with interpreted Java. The actual difference, therefore, is in reality much larger than the one shown.

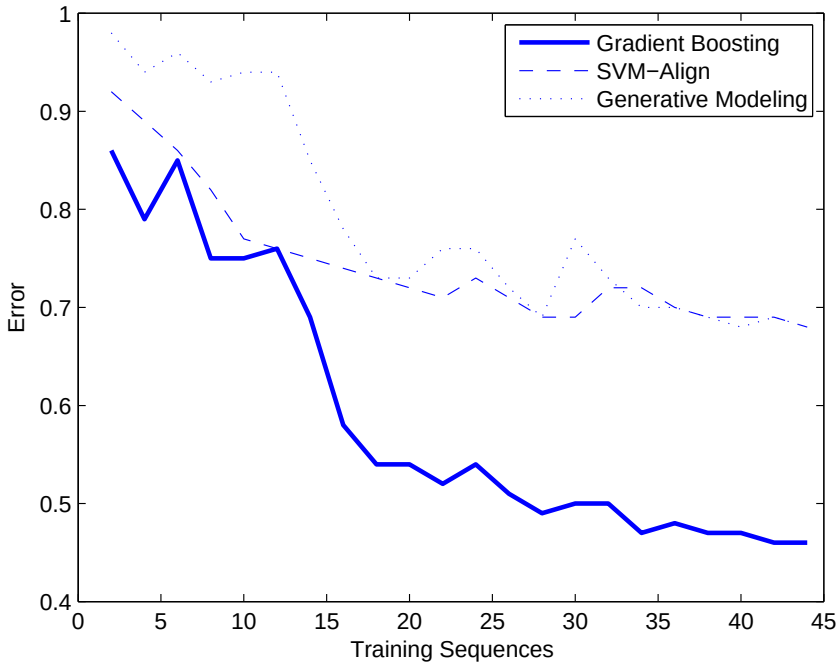


Figure 4.6: Accuracy boosting results in the synthetic domain.

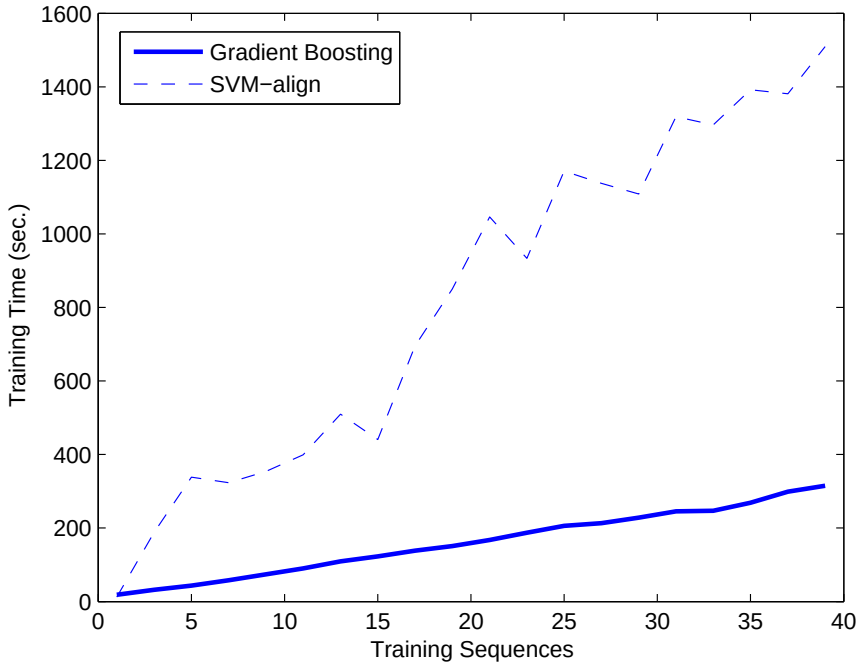


Figure 4.7: A running time comparison in the synthetic domain.

Chapter 5 – An Application to the Query-by-Humming Problem

There is a large and growing literature on the "query-by-humming" problem. In this problem, our set of target sequences is a set of songs (such as a private music collections, or all songs on all CD's available from amazon.com). Our query sequence is given by a user in some aural form. That is, the user attempts to render all or part of one of these songs by humming or whistling the song into a microphone connected to the computer. The goal is then for the computer to return a list of songs similar to the one sung by the user.

5.1 Domain Overview

There are several approaches for transforming the users sung data into a sequence of elements that can be matched to sequences in the target database. The foundation of all of them are the basic ideas of *pitch* and *time*. When a user sings a query, they sing a series of different frequencies, or pitches, as time elapses. Thus, each element, or note, in the query sequence must be at least a duple, one value for the pitch that the user is singing and another value for how long they sing this particular pitch.

Central to the parsing of the users query into a sequence of elements are the processes of *pitch recognition* and *note segmentation*. In pitch recognition, the

query audio is split into a number of "frames", each on the order of 0.01 seconds in length. The strongest pitch of that frame (ostensibly the one the user is trying to sing) is detected using frequency-domain auto-correlation [9] or time domain methods [31].

After each frame is assigned a pitch, consecutive frames of like pitch are often grouped together to form notes. This can be very difficult as there is usually a region of uncertain pitch between the end of one pitch and the beginning of the next. Many systems [74, 30, 44, 45, 66] have users sing syllables such as "TA" or "FA" rather than the actual lyrics of the song. The leading consonant introduces an artificial peak in the audio that is easy to detect and makes note segmentation trivial. Most other systems use hand-coded algorithms that work well on their datasets [57]. Some success has also been reported using neural networks for this task as well [59]. In Figure 5.1 we see a graphical depiction of the pitch detection and note segmentation processes.

Once the frames of audio have been assigned a pitch and grouped into notes, the data must be represented as a sequence for sequence matching. The structure of the elements of this sequence is not obvious and the choice of representation is crucial to the success of the system.

5.1.1 Pitch Representation

In terms of pitch, there are a variety of ways in which a user query can be represented. In many of the first query-by-humming studies [30, 44, 52] and even

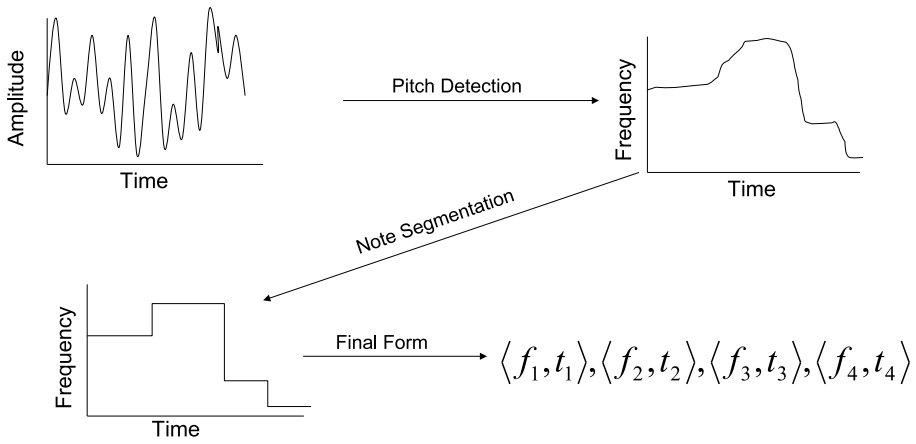


Figure 5.1: The processes of pitch detection and note segmentation.

some later ones [69, 73, 56] the representation of the sung query is *contour-based*: Rather than storing an absolute value for the pitch of each note, a *relative* value is stored. This value gives the positive or negative *interval* between the current note and the previous note. In this way, the query becomes *pitch invariant*. That is, because we only store relative values, the user may choose to start at any pitch so long as the relative values of the melody are maintained as it is rendered. The use of contour (relative values) to store melody has its roots in psychology [49, 24] and is generally accepted in the query-by-humming literature[1]. In addition, this has the advantage of avoiding the requirement that the users have *perfect pitch* (the ability to distinguish one pitch from another with no aural context), which is exceedingly rare.

The resolution at which the user melodies should be represented is also a point of contention. Because the pitch interval from one note to the next is real-valued, most previous systems "bin" this value in order to obtain a representation with a finite alphabet. Most of the early systems [74, 30, 52] relied on extremely coarse "UDR" representations, which only tell whether the current note is higher (U) lower (D) or at the same pitch (R) as the previous one. The initial thought was that most human singers could not render a melody with more precision than this three character alphabet could represent. However, several studies soon proved this contention incorrect [51, 66, 11] and several other studies showed that finer alphabets would make systems much more accurate [94, 89].

[1] A dissenting opinion is offered in [60]. Also note that there exist several systems that use absolute pitch [59, 26] as the melodic representation, so the idea of using relative pitch is by no means the last word.

In more recent systems, the alphabet size for contour has increased, first to five then to nine possible values [69, 56]. The most recent systems use a full semitone representation for contour, which gives a value for every key on the piano going one octave in either direction [66, 11]. This gives 27 possible values for the contour.

5.1.2 Time Representation

As with the pitch, the value for time is often stored as a value relative to the time of the previous note; in this case, the ratio between the current note and the previous note. Thus the representation of the song becomes not only pitch-invariant, but also *tempo-invariant*: The user no longer has to sing the song at a particular speed, so long as the relative durations of each note are maintained. While this has become the norm in most query-by-humming systems [66, 65] some systems choose to represent the time with absolute values instead [59, 52]. Furthermore, some studies suggest that tempo is fairly unreliable as a retrieval tool [65, 81], and so its representation may be a moot point.

Time resolution is also a matter of choice, as it was with pitch. However, far less study has been done regarding the proper resolution for the representation of time. One paper [65] presents reasonable evidence that a four-valued representation of time is all that is necessary for optimal discrimination between targets. I will assume that this resolution is optimal in what follows.

Aside from the choice of representing time with absolute or relative values, there are a number of exotic representations that bear mentioning. In [45], a tempo is

clicked as a user sings the melody, and time is measured as the number of clicks per note, thus giving rise to a *beat-based* representation of the time for each note. In [54] and [64], a *frame-based* representation is used. In this representation, the note segmentation step is done away with altogether and the matching process works with the raw, pitch-detected frames of audio. While this does away with the often error prone step of segmenting the frames, it causes massively slower running times [38] and apparently not much increase in performance [18].

5.1.3 Target Processing

The process of transcribing the target into a sequence of notes is even harder. Typically a target song is *polyphonic*, meaning that there are often many more notes happening at once than just the main sung melody. The song is also much longer than any user is willing to sing: In a typical query, only a small, memorable part of the song is rendered.

This means that there are several difficult subproblems within the problem of target transcription. Among these are *polyphonic transcription* (transcribing a raw audio signal that is polyphonic), *melody spotting* (distinguishing melody notes from non-melody ones), and *thematic extraction* (extracting coherent themes from a long melody sequence). Each of these problems is highly complex and has an entire literature of its own.

For our experiments, in order to isolate the part of the process in which I am interested, I elect to use a preprocessed midi database of 2000 targets known as

the *Digital Tradition Folk Song Database* [23]. This is a collection of American folk songs stored in monophonic midi format, and they are thus in exactly the appropriate form for sequence matching.

In Figure 5.2, as a summary, I give a possible form of a complete query by humming system. The left branch of this figure represents the processing pipeline for the target data and the right branch represents the processing pipeline for the query data. By the time the branches meet (when an actual query is given to the system) we have symbolic representations of both the target and query songs.

5.1.4 Experimental System

I now describe the exact query-by-humming representation used in the experiments that follow in Chapters 4 and 6.

The user query is transcribed as described above. Both pitch detection and note segmentation are done on the raw audio. When transcription is complete, the events in the query sequence are represented as a series of tuples, containing a component for the *average pitch* and the *duration* of each event, so a query sequence $\mathbf{s}$ is of the form:

$$\mathbf{s} = \{(s_1^p, s_1^d), (s_2^p, s_2^d), \ldots, (s_{|\mathbf{s}|}^p, s_{|\mathbf{s}|}^d)\}$$

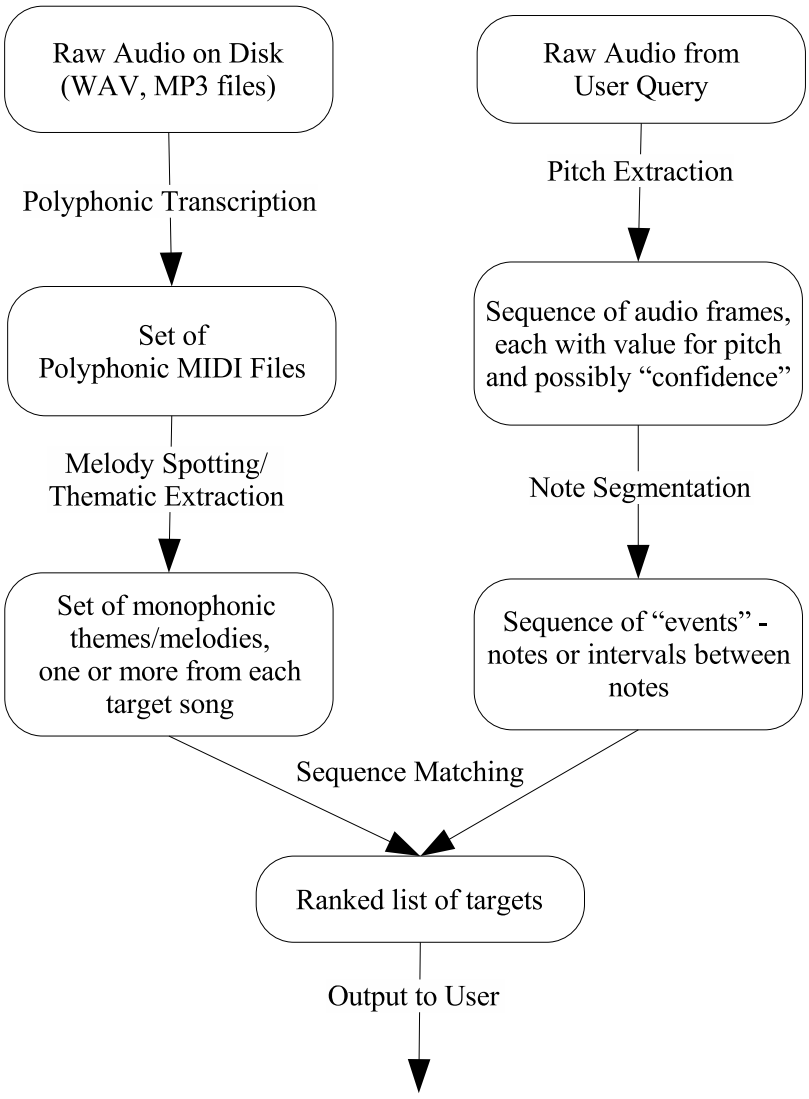


Figure 5.2: A possible general form of a Query-by-Humming system.

The extracted sequence is then further processed into a relative representation, using *pitch differences* and *duration ratios*:

$$\mathbf{s} = \{(s_1^\delta, s_1^r), (s_2^\delta, s_2^r), \ldots, (s_{|\mathbf{s}|}^\delta, s_{|\mathbf{s}|}^r)\}$$

where $s_i^\delta = s_{i+1}^p - s_i^p$ and $s_i^r = \frac{s_{i+1}^d}{s_i^d}$.

The query audio I use for these experiments is a body of 587 queries gathered from several amateur choir singers (participating in church and college choirs), as well as several non-singers. I chose amateur singers because they are the most likely users of a finished query-by-humming system. There were 50 singers in all, and there were 12 query songs, all of which were played for the user on a piano before they rendered their query, and all of which were generally well-known by the users.

5.2 Experimental Results

In this first of set of experiments, I attempt to learn an accurate distance function using the process given in Figure 4.5. I compare the performance of gradient boosting with the generative method and SVM-align on training sets of similar size. In these experiments, the alphabet of notes for the query-by-humming domain is made finite for the purposes of SVM-align and generative modeling. Using values suggested in the literature [11, 65] I use 27 values for pitch and 4 values for duration giving $27 \times 4 = 108$ characters in the alphabet. For the gradient

boosting method, no binning of the notes is done; they remain real-valued vectors throughout training. Testing occurs on small targets sets of only 50 sequences, so that we can more closely study the effects of sparse training data on candidate algorithms. Regression trees are trained to minimize squared error on the training set.

5.2.1 Comparison to Other Methods

For the learning done below, the correct "training alignments" are provided by the note-interval alignment function given in [18]. In this work, that alignment function is shown to be on par with a complex graphical model and a time-consuming frame-based method in terms of retrieval accuracy. I will then assume in what follows that this function is the state-of-the-art.

In Figure 5.3 we see the learning curves in the query-by-humming domain. It is clear that both gradient boosting and SVM-align are able to benefit from learning in a discriminative way, as both methods are able to progress much faster to a reasonably accurate function than the generative method. Gradient boosting only appears to have a slight performance advantage in this domain. I believe this is due to the fact that the number of query songs is fairly small and thus, even with cross-validation, small training sets can still provide exhaustive information about the test data (that is, all of the characters in the alphabet that are seen in test are also seen in training). This means that the chief advantage of gradient boosting, its

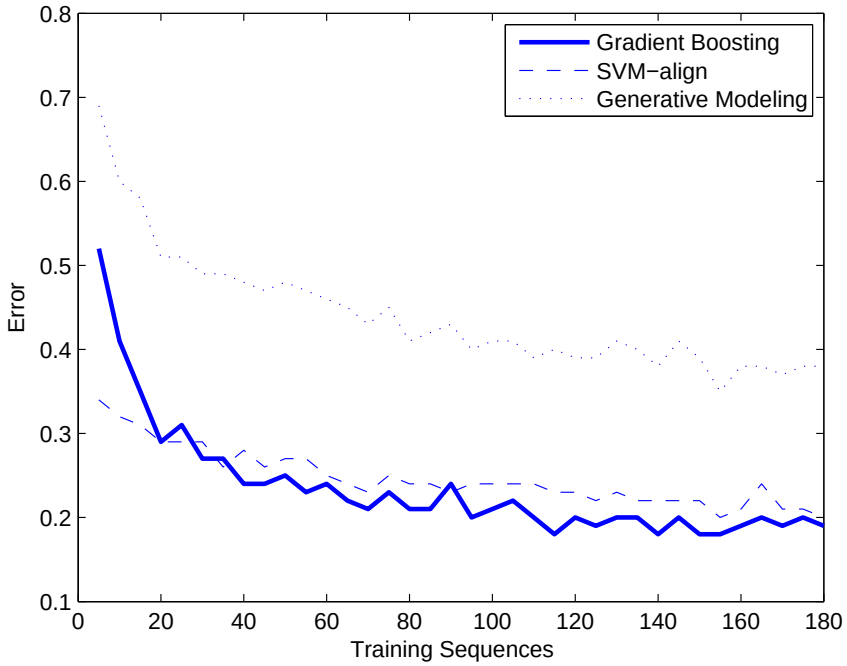


Figure 5.3: Accuracy boosting results in the query-by-humming domain.

ability to generalize cost behavior to points unseen in the training data, is rendered largely useless in this domain.

Nonetheless, structured gradient boosting is able to match the performance of the much more computationally intensive SVM-align approach and handily outperforms the generative approach, apparently even without exploiting its representational advantages.

5.2.2 Boosting The State-of-the-Art

It was mentioned earlier that the initial cost function given to the accuracy boosting algorithm could be either hand-designed or uninformative. In all of the previous experiments shown, the costs were initialized to random values. I now do an experiment using the function from [18] in the query-by-humming domain that I have used in the previous experiments to generate training alignments. I do 30 iterations of accuracy boosting on this function, using alignments that it itself has generated.

The final plot shows *mean reciprocal rank* versus boosting iteration. I use it for this final plot because it is a benchmark used often in the literature on this subject [18, 81] and so offers the most objective point of comparison for the hand-coded function and its boosted counterpart. It is computed by averaging the reciprocal of the rank of the correct target as computed by the scoring function for all queries $\mathbf{q}_i$. If there are k queries in the test set:

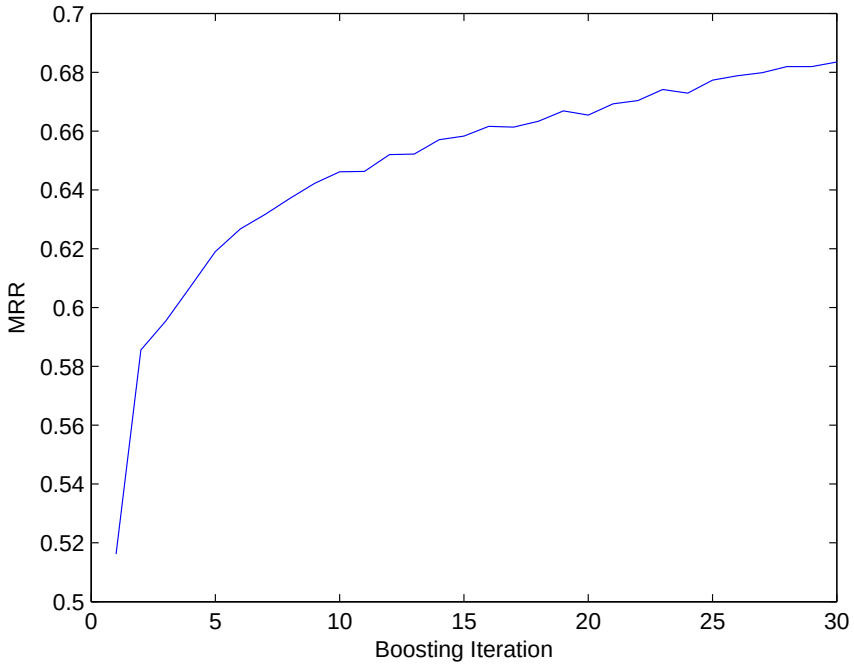


Figure 5.4: Accuracy boosting with the state-of-the-art distance function in the query-by-humming domain.

$$\text{MRR} = \frac{\sum_{i=1}^{k} \frac{1}{\text{rank of } \mathbf{t}_i}}{k} \tag{5.1}$$

In Figure 5.4 we see the results of that experiment. The MRR of the function after boosting is .15-.20 higher than the original function. This result is one of the most major of this work. The function I am boosting here was determined in [18] to be the state of the art, and I am able to improve its performance with these techniques by a substantial amount. This ability to incorporate and build on

human-given costs is an extremely useful feature of this algorithm, and one that SVM-align lacks without substantial re-engineering.

Chapter 6 – Learning Efficient Distance Functions

We have thus far seen how to use structured gradient boosting to learn distance functions that perform with high accuracy. However, finding the correct target structure given a query structure still requires computing the distance function from the query to every target in the target set. These computations may be complex and not practical for even moderately large target sets [18]. Moreover, the matching of the query to every possible target in the target set seems intuitively unnecessary.

In this chapter, I show how structured gradient boosting can be used to tune the cost functions for improved retrieval performance using metric access methods [36]. In particular, I show how to construct a data structure for efficient retrieval in a database of sequences, and then how to tune the cost function, using structured gradient boosting, to improve the performance of this data structure.

6.1 Metric Distance Functions and Metric Access Methods

Metric access methods attempt to alleviate the need to search the entire target set to reach the correct target for the given query. Essentially, all of these methods construct a tree or graph over the target set using the distance function. The tree is structured so that the distance from a query to certain elements in the target set

implies that certain other elements need not be considered in the search. Figure 6.2 shows an example of such a tree. We will revisit the tree in more detail later.

Although the details of this structure vary from method to method, these trees all rely on certain properties of the distance function to prune the space of possible nearest neighbors. Specifically, all metric access methods require that the distance function be *metric*. This means it must satisfy four criteria for all x, y, and z in the domain of possible structures $\mathcal{D}$:

$$x \neq y \Rightarrow d(x, y) > 0 \qquad \text{(positivity)} \tag{6.1}$$

$$x = y \Rightarrow d(x, y) = 0 \qquad \text{(identity)} \tag{6.2}$$

$$d(x, y) = d(y, x) \qquad \text{(symmetry)} \tag{6.3}$$

$$d(x, y) + d(y, z) \geq d(x, z) \qquad \text{(triangular inequality)} \tag{6.4}$$

The first three can be satisfied with relative ease in the sequence alignment domains. Positivity can be enforced by taking logarithms of probabilities in the generative method, and by ensuring that no $c_k(a, b)$ ever gets within some ϵ of zero in gradient boosting, and in both cases reversing the sign. Identity can be enforced by returning 0 if the arguments are equal. Symmetry can be enforced by creating a slightly redefined distance function d^* from the original d, such that $d^*(x, y) = \min(d(x, y), d(y, x))$, or failing this, in other, more domain-dependent ways. Though these enforcements restrict the expressiveness of the distance func-

tion, we will see that it is still possible to learn a highly accurate distance function under such restrictions.

The triangular inequality, however, is far more difficult. There is no obvious way to ensure that a learned distance function will satisfy the triangular inequality. Recent work, however, aims at modifying a distance function so that it satisfies the triangular inequality with high probability. It is this work that I review next.

6.2 Enforcing the Triangular Inequality

First, observe that if the triangular inequality fails on some three points it is because the distance between some pair of these is greater than the sum of the distances between the other two pairs. To "repair" this triple so that the triangular inequality is satisfied, we need only close the gap between these distances. If we can modify distance function d in a way that maintains the ordering on all distance computations, but repairs all broken triples, we will have maintained the retrieval accuracy and transformed d into a metric.

The insight of [86] is that the simple application of any concave function g to the computed distances will do exactly this: Since g is monotonic, it insures that the computed distances maintain their ordering. Also, since g is concave, the distances between elements in the new metric space move closer together, so the triangular inequality is satisfied in more triples.

The caveat to this is that if g has too high of a degree of concavity, then all distances as measured by d become nearly the same. Because all metric access

methods rely on some distances being far greater than others to do efficient search, this makes them useless in the context of d and we are back where we started. We would like then, to have a function g with concavity sufficient to repair all non-triangular triples, but no more.

Fortunately, there is a simple function with a tunable concavity:

$$g(x) = x^{\frac{1}{1+w}} \tag{6.5}$$

If we apply this to d, we have a new distance metric.

$$d_g(x, y) = g(d(x, y)) = d(x, y)^{\frac{1}{1+w}} \tag{6.6}$$

A plot of this new function in relation to the original distance function is shown in Figure 6.1. As $w \to \infty$, the function reaches maximum concavity (a right angle at $(0, 1)$) and as $w \to 0$ there is no concavity at all. Thus, we do a simple line search of w to find the point at which the triangular inequality is "sufficiently" satisfied to use metric access methods. To check how well a given d_g satisfies the triangle inequality, we can sample triples from the target set $\mathcal{T}$. Empirically, the experiments at the end of the chapter show that if the triangular inequality is satisfied with $P > 0.99$, the errors encountered when using metric access methods will be negligible, but this is domain-specific.

The astute reader will note that there is no guarantee that a given d will respond well to these methods. In particular, it is possible that creating a near-metric from d requires g to have concavity so high that d_g will be completely incompatible with

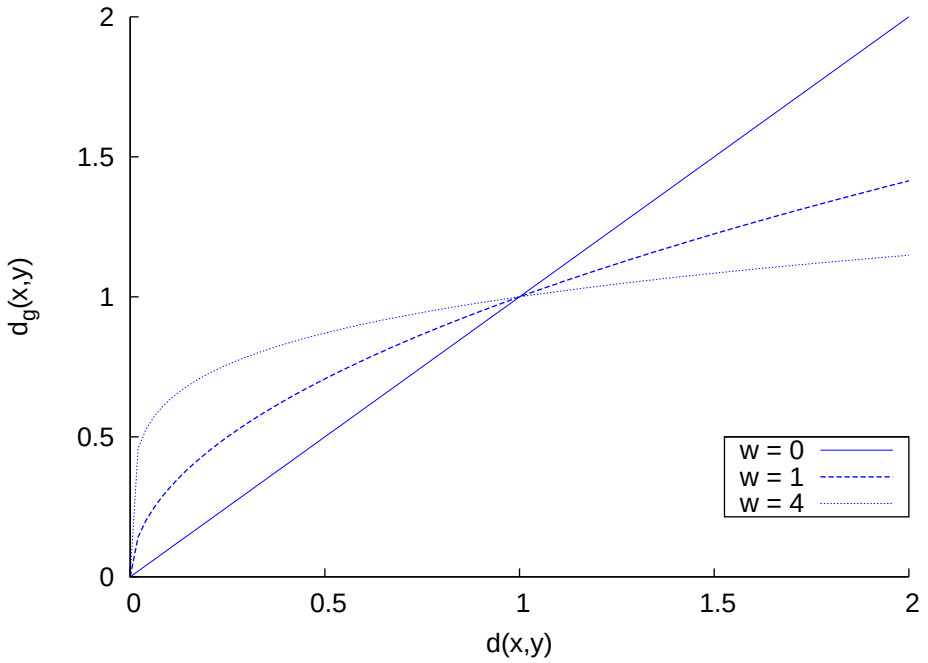


Figure 6.1: A plot of the function $d_g(x, y) = d(x, y)^{\frac{1}{1+w}}$ at various values of w.

metric access methods. The experiments in [86] show, however, that this is not the case in many domains.

Thus, it is possible to create a near-metric distance function from a distance function learned by the methods above, though its usefulness must be determined empirically. I will now discuss a metric access method, the *vp-tree*, and show how a near-metric distance function can be optimized to a particular instance of this metric access method using gradient boosting.

6.3 The vp-tree

The structure I use for these experiments is the *vantage point tree* or *vp-tree* [36]. I choose the vp-tree due to its relative simplicity and straightforward application, but these techniques may be easily applied to other metric access methods. This is discussed in the conclusion.

Each node in the vp-tree is defined by a chosen target structure from the target set, the vantage point. If there is only a single target in the target set, this target is the vantage point and nothing more need be done. If there is more than one, the targets in the target set are divided into two nearly equal subsets based on the distance from the vantage point. The left child of the given node is then constructed recursively using the subset of targets less than the median distance from the vantage point, and the right child is similarly constructed from the other subset.

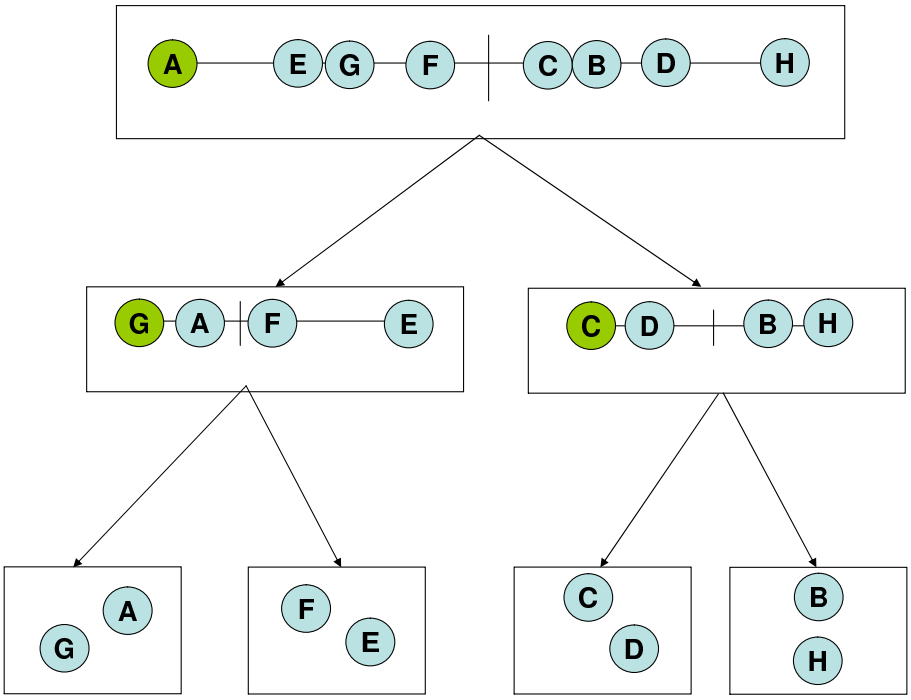


Figure 6.2: A vantage point tree. Each circle represents an element of the target set. The rectangles are nodes in the tree, with the dark colored circles at the left of each rectangle representing the vantage point of each node.

Figure 6.3: An algorithm for constructing a vp-tree.

$\mathcal{T}$ is a set of target structures, $\mathbf{t}$ is a target structure, $\mathbf{n}$ is a node in the tree with children $\mathbf{n}_l$ and $\mathbf{n}_r$, and d is a metric distance function. $m_{\mathbf{n}}$ is the median score at node $\mathbf{n}$ and $\mathbf{t}_{\mathbf{n}}$ is the vantage point.

```
function CONSTRUCT-TREE(n, T, d)
  s ← |T|
  if s = 1 then
    t_n ← T[0]
    return
  end if
  t_n ← CHOOSE-VANTAGE-POINT(T)
  sort T on d(t_n, t_i ∈ T)
  m_n ← d(t_v, T[s/2])
  CONSTRUCT-TREE(n_l, T[0 : s/2], d)
  CONSTRUCT-TREE(n_r, T[s/2 : s], d)
end function
```

We see a graphical representation of the first three levels of a vp-tree in Figure 6.2. Each circle represents a sequence in the target set and each of the rectangle enclosing the circles represents a node in the tree. The darker sequences (sequence "A" in the root node) are the vantage points associated with each node. As we can see, the other sequences are ordered by their distance to the vantage point and the set is split into right and left subtrees based on the median distance. To complete the construction, the bottom set of nodes in Figure 6.2 should be split again into leaf nodes containing a single sequence and having no children. Algorithm 6.3 shows the construction algorithm in pseudocode.

To see how we can leverage the properties of a metric distance to speed up search, we first consider the following definition:

$$d_{\mathbf{v}}(\mathbf{t}, \mathbf{q}) = |d(\mathbf{t}, \mathbf{v}) - d(\mathbf{q}, \mathbf{v})| \tag{6.7}$$

The metric properties of symmetry and the triangular inequality gives:

$$d(\mathbf{t}, \mathbf{q}) \geq |d(\mathbf{t}, \mathbf{v}) - d(\mathbf{q}, \mathbf{v})| = d_{\mathbf{v}}(\mathbf{t}, \mathbf{q}) \tag{6.8}$$

so that distances get smaller when measured by $d_{\mathbf{v}}$ as opposed to d. It follows that:

$$d_{\mathbf{v}}(\mathbf{t}, \mathbf{q}) \geq \tau \Rightarrow d(\mathbf{t}, \mathbf{q}) \geq \tau \tag{6.9}$$

Suppose we wish to find the nearest neighbor of a query $\mathbf{q}$ within range τ, and we are at a node $\mathbf{n}$ in the tree with vantage point $\mathbf{v}$. Let the median distance between $\mathbf{v}$ and all targets under $\mathbf{n}$ be $m_{\mathbf{n}}$. Hence $d(\mathbf{t}, \mathbf{v}) < m_{\mathbf{n}}$ for all targets $\mathbf{t}$ in the left subtree. If $d(\mathbf{q}, \mathbf{v}) \geq m_{\mathbf{n}} + \tau$, then it follows from Equations 6.7 and 6.9 that the left subtree of $\mathbf{n}$ may be eliminated from consideration. Similarly, if $d(\mathbf{q}, \mathbf{v}) < m_{\mathbf{n}} - \tau$ then we may eliminate the right subtree. However, if $m_{\mathbf{n}} + \tau > d(\mathbf{q}, \mathbf{v}) \geq m_{\mathbf{n}} - \tau$ then we must do a linear search of all descendants of $\mathbf{n}$. Figure 6.4 shows this recursive algorithm in psudeocode. Note that we can get the nearest neighbor in the entire tree by calling the algorithm on the root node.

We see from these two algorithms that the choice of the vantage point is crucial to the success of the tree. Normally, we would like to choose a vantage point that projects the target set into a space where few of the distances are close to the median distance. In this work, however, I take a different approach: Given a

Figure 6.4: An algorithm for retrieval in a vp-tree.

$\mathbf{q}$ is a query structure for which we want to find the closest neighbor within distance τ in the tree. $\mathbf{n}$ is a node in the vp-tree, where $m_{\mathbf{n}}$ is the median score at $\mathbf{n}$ and $\mathbf{t_n}$ is the vantage point. $\mathbf{n}$ has child nodes $\mathbf{n}_l$ and $\mathbf{n}_r$ (if they exist). d is a metric distance function. The function SEARCH-ALL-CHILDREN searches all descendants of argument node for the closest neighbor to the argument query under the argument distance.

```
function GET-NEAREST(q, n, τ, d)
  if n is a leaf node then
    return t_n
  end if
  if d(t, q) < n_m − τ then
    return GET-NEAREST(q, n_l, τ, d)
  end if
  if d(t, q) > n_m + τ then
    return GET-NEAREST(q, n_r, τ, d)
  end if
  return SEARCH-ALL-CHILDREN(q, n, d)
end function
```

constructed tree and the associated vantage points, I will modify the distance function to obtain better performance from the given tree. More specifically, I will define a notion of loss and margin associated with querying the constructed tree, then compute functional gradients that modify the distance function to decrease the loss and increase the margin.

6.4 Adapting Gradient Boosting for Efficiency Learning

We present here the algorithm originally presented in [68]. This algorithm computes the functional gradients against the constructed vp-tree in much the same way as was done in Chapter 4: For each training query, the algorithm considers each event from the best alignment of each non-leaf target on the query's path from root to leaf. The intuitive direction of the gradient is obvious from the construction of the vp-tree: at each node in the path, the query score should be moved as far as possible to the correct side of the median, thereby allowing the search to progress down the tree even at high values of τ. The margin for each query and path node, then, is the amount by which it lies on the correct side of the median. Loss is incurred when this margin is negative, and the gradient will attempt to make this margin as large as possible.

This loss, however, is not uniform for each node on the path: Recall that being within τ of the median at a particular node $\mathbf{n}$ forces us to abandon the tree-search and perform a linear scan of all leaves that are descendants of $\mathbf{n}$. If $\mathbf{n}$ is one level above the leaf node, the computational cost of being within τ at $\mathbf{n}$ is a single

extra distance computation. However, if $\mathbf{n}$ is the root, the search fails to eliminate a subtree containing $|\mathcal{T}|/2$ targets and the computational cost increases on that order. More specifically, suppose there are targets $\{\mathbf{t}_1, \mathbf{t}_2, \dots\}$ on a path from root to leaf for a given query, where $\mathbf{t}_1$ is the root. Hence, the training loss at $\mathbf{t}_j$ is weighted by a factor of $\frac{1}{2^j}$, as the vp-tree construction algorithm assures a balanced tree. This can be incorporated into the objective directly by rescaling the training loss as described in Section 2.4.1.

After the gradient is computed, it is added to the current distance function and the process is repeated. However, there is the possibility that this modification of the distance function has invalidated the tree, either by gross violation of the triangular inequality or by changing the distance between targets such that some are now in the incorrect subtrees of their parent nodes. These situations can be remedied at each step by applying the triangle-generating procedure of Section 6.2, and by rebuilding the tree where it is incorrect.

6.4.1 Computing the Functional Gradients

More formally, consider a training set $\mathcal{S}$ and constructed vp-tree V with queries $\{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_{|S|}\}$. Each of these queries has a single path $\{\mathbf{n}_{i1}, \mathbf{n}_{i2}, \dots, \mathbf{n}_{ip}\} \in V$ from root to the leaf containing the correct target, where p is the number of nodes in the path. Each node in this path has an associated target as its vantage point. Call the targets associated with the path of $\mathbf{q}_i$ $\{\mathbf{t}_{i1}, \mathbf{t}_{i2}, \dots, \mathbf{t}_{ip}\}$.

I now compute the margin for each pair $(\mathbf{t}_{ij}, \mathbf{q}_i)$, defined as the difference between median distance to $\mathbf{t}_{ij}$ and $d(\mathbf{t}_{ij}, \mathbf{q}_i)$. The larger this distance is, the higher the chance of conducting an effective search. If m_{ij} is the median distance to $\mathbf{t}_{ij}$ then the margin m_i is

$$m_i = m_{ij} - d(\mathbf{t}_{ij}, \mathbf{q}_i) \tag{6.10}$$

if the correct target given $\mathbf{q}_i$ is in the left subtree of $\mathbf{n}_{ij}$ and negation of the same expression if it is in the right subtree. For convenience, define a value v_{ij} such that if the correct target for $\mathbf{q}_i$ is $\mathbf{t}_i$ as in Chapter 4, then:

$$v_{ij} = \begin{cases} 1 & \text{if } \mathbf{t}_i \in \text{ left subtree of } \mathbf{n}_{ij} \\ -1 & \text{otherwise} \end{cases} \tag{6.11}$$

Stating the margin in a single equation gives:

$$m_i = v_{ij}(m_{ij} - d(\mathbf{t}_{ij}, \mathbf{q}_i)) \tag{6.12}$$

Again, I borrow the following loss formulation from LogitBoost:

$$\log(1 + \exp(v_{ij}(d(\mathbf{t}_{ij}, \mathbf{q}_i) - m_{ij}))) \tag{6.13}$$

Recall that d in the sequential alignment setting is simply the sum of the costs of the events in the optimal alignment. Recall also from Chapter 4 that $f(a, b, \mathbf{x}_i, \mathbf{y}_i)$ is the number of times that character a is replaced by b in the optimal alignment of sequences $\mathbf{x}_i$ and $\mathbf{y}_i$. Define the following shorthand:

$$f_{ij}(a,b) = f(a,b,\mathbf{t}_{ij},\mathbf{q}_i) \tag{6.14}$$

With this definition, the distance function is:

$$d(\mathbf{t}_{ij},\mathbf{q}_i) = \sum_a \sum_b c(a,b) f_{ij}(a,b) \tag{6.15}$$

The loss L for each query is in a sum of the losses at each target in the path to the correct leaf. In addition, I use a misclassification cost function of $\frac{1}{2^j}$ and rescale the margin-based loss, as discussed earlier in this section.

$$L = \sum_j \frac{1}{2^j} \log(1 + \exp(v_{ij}(d(\mathbf{t}_{ij},\mathbf{q}_i) - m_{ij})) \tag{6.16}$$

I now perform the crucial step. The loss function is derived with respect to the current scoring function c_k at each pair (a,b). This gives the gradient step $\delta_{k+1}(a,b)$:

$$\nabla L_{k+1}(a,b) = \frac{\partial L}{\partial c_k(a,b)} \tag{6.17}$$

$$= \sum_j \frac{v_{ij} f_{ij}(a,b) \exp(v_{ij}(d(\mathbf{t}_{ij},\mathbf{q}_i) - m_{ij}))}{2^j(1 + \exp(v_{ij}(d(\mathbf{t}_{ij},\mathbf{q}_i) - m_{ij})))} \tag{6.18}$$

Simplifying the above expression yields the final functional gradient expression[1]:

$$\nabla L_{k+1}(a,b) = \sum_j \frac{v_{ij} f_{ij}(a,b)}{2^j(1+\exp(v_{ij}(m_{ij} - d(\mathbf{t}_{ij}, \mathbf{q}_i))))} \tag{6.19}$$

6.4.2 Ensuring the Validity of the vp-tree

To ensure the validity of the constructed vp-tree as changes are made to d, the learning process must check each node to make sure the two child vantage points are still on the appropriate sides of the median value. This is done by passing a target set into the node (as in construction), and checking to make sure that the child vantage points are in this set and on the correct side of the median distance. If they are, the appropriate halves of the target set are sent to the right and left child nodes and the check performed recursively. If not, the subtree starting at the failed check must be rebuilt.

6.5 Experimental Results

In this set of experiments, I attempt to learn a function that retrieves efficiently, but also with high accuracy. For this I use a two-phase boosting approach, in which an accurate distance function is learned in the first phase, followed by construction of a vp-tree over the learned distance function, and then a second boosting phase to optimize the efficiency of this tree. In this second phase, I use a weighted

[1]Remember from Equation 4.16 the distance function is computed as the *negation* of the sum of the feature values. As such, the direction of the gradient step will need to be reversed after it is computed.

combination of the accuracy and efficiency gradients to assure that accuracy is maintained as efficiency is boosted. This is compared to using only accuracy boosting throughout both iterations. Target sets of 2000 sequences are used for training and testing. The vantage points of the vp-tree are selected in the standard way by taking a random sample of 20 targets at each node in the tree as it is constructed, computing the distance to each of the targets from all other targets at the current node, and choosing the target for which the computed distances have the highest variance.

Figure 6.5 shows the general form of the two-phase algorithm in psudeocode. Note that in the first half of the algorithm, a regression tree approximating the gradient vector, ∇L, is learned, as in Chapter 4. In the second half of the algorithm, however, ∇L is discritized and the results counted into a table. This provides a large, sparse vector representing the counts in the training set. The sparsity of the vector is crucial because the goal is to specialize the distance metric to the constructed tree. A few brief experiments showed that learning an approximation effectively defeats this specialization, creating updates to the cost function that are too broad to have effectiveness.

In all of the experiments below, ten-fold cross validation was used to minimize the effects of randomness. The learning curves all plot error against training set size. The accuracy curves for efficiency boosting plot error against boosting iteration to show the possible degradation in accuracy performance when the switch is made from accuracy boosting to efficiency boosting.

Figure 6.5: The two phase boosting algorithm

$\mathcal{T}$ is a set of target structures, $\mathcal{Q}$ is a training set of query sequences (for which correct targets are known), and c_k is the set of costs at iteration k. ∇L is a vector representing the functional gradient. n_a and n_e are the numbers of boosting iterations in the accuracy and efficiency phases of the algorithm, respectively, and β is a parameter that specifies the relative importance of accuracy maintenance and efficiency improvement in the efficiency boosting phase. Note that c_k, in combination with the Smith-Waterman algorithm, specifies a metric distance function that can be used in CONSTRUCT-TREE.

function BOOST-TWO-PHASE($\mathcal{Q}, \mathcal{T}, c_k, \alpha$)
  for $k = 0$ to n_a **do**
    $\nabla L_{k+1}(a, b) = 0, \forall(a, b)$
    for all $\mathbf{q}_i \in \mathcal{Q}$ **do**
      $\nabla L_{k+1} \leftarrow \nabla L_{k+1}$ + COMPUTE-ACC-GRADIENT($\mathbf{q}_i, \mathcal{T}, c_k$)
    end for
    $\mathcal{S} = \{((a, b), \nabla L_{k+1}(a, b))\}, \forall a, b$
    $T_{k+1} \leftarrow$ LEARN-REGRESSION-TREE($\mathcal{S}$)
    $c_k \leftarrow c_k + T_{k+1}$
  end for$M =$CONSTRUCT-TREE(root, $\mathcal{T}, c_k$)
  for $k = 0$ to n_e **do**
    $\nabla L_{k+1}(a, b) = 0, \forall(a, b)$
    for all $\mathbf{q}_i \in \mathcal{Q}$ **do**
      $\nabla L_{k+1} \leftarrow \nabla L_{k+1} + (\beta)$COMPUTE-ACC-GRADIENT($\mathbf{q}_i, \mathcal{T}, c_k$)
      $\nabla L_{k+1} \leftarrow \nabla L_{k+1} + (1 - \beta)$COMPUTE-EFF-GRADIENT($\mathbf{q}_i, \mathcal{T}, c_k, M$)
    end for
    $c_k \leftarrow c_k + \alpha \nabla L_{k+1}$
    VALIDATE-TREE(M, c_k)
  end for
end function

The efficiency curves plot the number of targets not directly compared to the query against the error tolerance. Obviously, the higher the error tolerance, the more targets will be pruned from the target set before explicit consideration and the more efficient the function will be. The curve, then, is much in the spirit of an ROC curve, where the best result is to have nearly all of the targets pruned even at low error tolerances.

We first look at the results in the query-by-humming domain. We see in Figure 6.6 the accuracy of the function as the two-phase boosting approach progresses. As we would expect, there is a brief degradation in accuracy as efficiency boosting begins at the 30th iteration. This is only temporary, however, and the accuracy of the learned function returns to within 1% of its pre-efficiency boosted accuracy by the time efficiency boosting is completed.

In Figure 6.7, we see the results of efficiency boosting in this domain. As we can see, the learning process is able to achieve efficiency advantages over accuracy only boosting of up to 10% of the target set at the crucial low error rates. As we can also see, it seems that this domain is already fairly receptive to metric access methods, even without efficiency boosting. It is encouraging that, even in the case where metric access methods are already applied with some success, efficiency boosting is able to provide retrieval speed up with only small loss of accuracy.

The next set of figures shows the same set of experiments in the synthetic domain. Here, the results are even more definitive. Figure 6.8 shows only a brief hiccup in accuracy as the function is boosted, resulting in no performance loss in

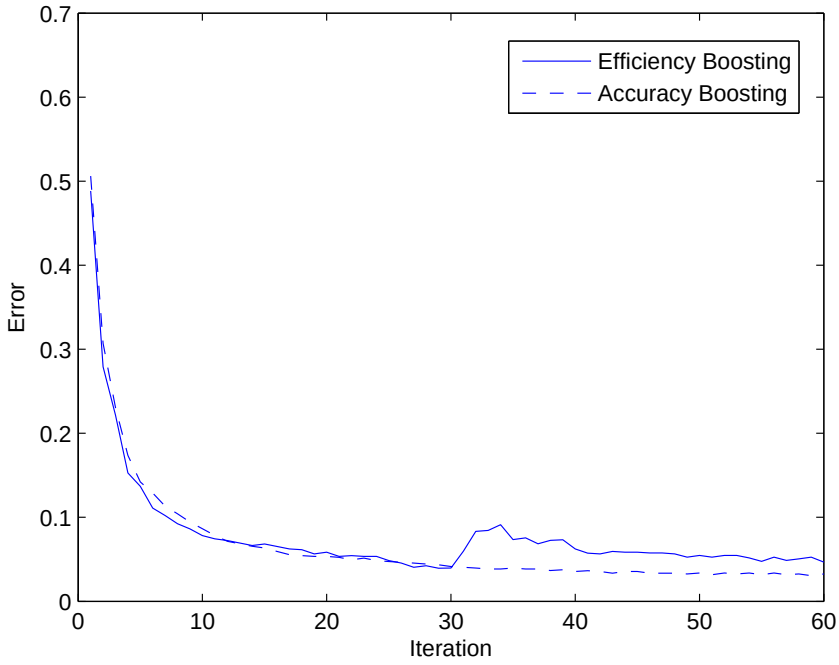


Figure 6.6: Accuracy with the two-phase boosting approach in the query-by-humming domain.

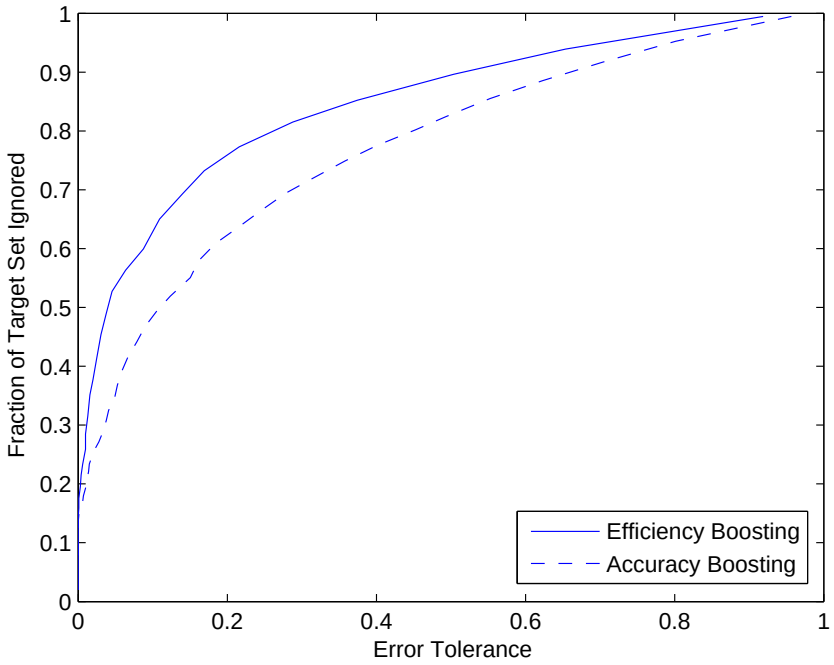


Figure 6.7: Efficiency plot in the query-by-humming domain.

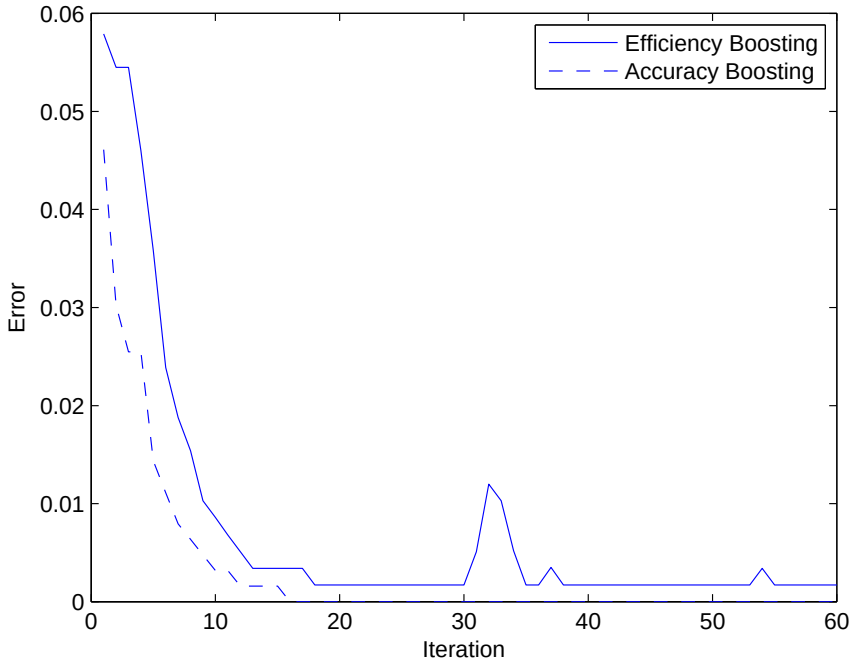


Figure 6.8: Accuracy with the two-phase boosting approach in the synthetic domain.

the long run. Note that the scale of the y-axis is so small that the difference in error between the accuracy and efficiency boosted functions is almost meaningless.

Figure 6.9 shows a significant increase in efficiency in this domain; nearly 30% at low error rates. Perhaps more interestingly, the initial, accuracy-only function is not at all receptive to metric access methods: the probability that a target will be pruned is equal to the probability that this pruning will lead to an error, resulting in the straight line of the accuracy boosting curve. Efficiency boosting is able to make metric access methods at least somewhat useful in this case.

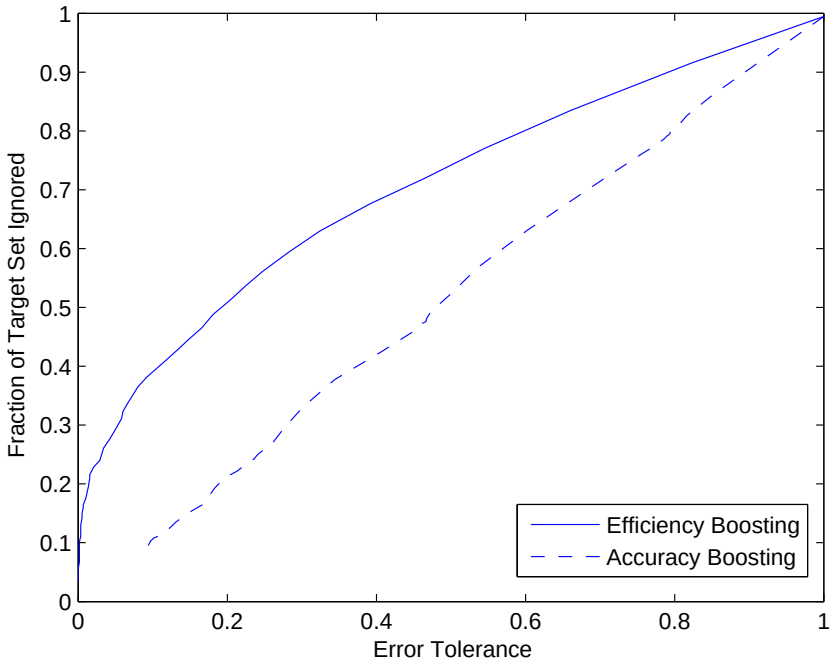


Figure 6.9: Efficiency plot in the synthetic domain.

Chapter 7 – Conclusion

Herein, I have discussed some of the methods related to a new facet of the machine learning literature known as structured prediction. In particular, I have discussed two structured prediction algorithms in detail, structured perceptron and structural support vector machines, and attempted to consolidate some of their strengths into a new algorithm, structured gradient boosting. I showed that this algorithm is able to incorporate general notions of classification loss and can be efficiently kernelized.

I then showed how this algorithm can be applied in a simple way to the learning by demonstration paradigm, by treating starting state as the input domain and possible plans as the output domain. I showed results in a real-time strategy game domain that indicate that complex objective functions can be learned with structured gradient boosting using little training data, simple features, and a linear model.

Following this, I presented a comprehensive approach to learning distance functions for sequence alignment via structured gradient boosting; an approach that not only learns distance functions that are highly accurate, but also an approach that can contribute to efficiency of retrieval. The approach, in brief, learns the distance function iteratively. At each iteration, we compute an accuracy gradient based on the discriminative nature of the retrieval problem and a suitable notion of

margin. I have shown empirically that in at least two sequence retrieval domains, this approach is able to learn distance functions that are more accurate than traditional generative approaches, and also at least as accurate if not more accurate than SVM-align [92], a recently introduced and very promising discriminative approach.

As stated above, this approach is also able to increase the efficiency of a learned distance function. This is accomplished by modifying the gradient above to iteratively improve performance on a given *metric access method* (in this case a vp-tree). Experimental results show that the approach is able to improve efficiency of vp-trees in these domains by 10-30%, even at relatively modest target set sizes.

Finally, I have shown that accuracy boosting is able to to improve the accuracy of a hand-coded function in the query-by-humming domain. The function improved by learning was shown to be state-of-the-art in [18] and underwent significant improvement (from an MRR of 0.51 to an MRR of 0.68) after accuracy boosting.

The success of the accuracy boosting technique seems to depend on two facts: First, the approach is *discriminative* rather than *generative* in nature. This enables features relevant to proper discrimination to be identified quickly and given substantial importance, thus learning a reasonably accurate function with less training data than is required by traditional generative learning, which requires ad-hoc assumptions about the feature space. Second, my approach has a representational advantage: Instead of treating sequence elements as atomic characters with no internal structure or relationship, as in previous methods, I utilize the structure and interdependence of the sequence alphabet to learn the cost behavior of sequence

elements that did not appear in the training data. This is accomplished by using a regression tree to approximate the computed gradient in each iteration of the boosting algorithm.

Just as discriminative learning is able to, in some sense, encode the target database into the distance function, the efficiency gradient finds its success by encoding a constructed vp-tree into the distance function. As the efficiency gradient progresses, it constructs cost modifications that cause query sequences to travel down the correct path in the constructed tree, especially focusing on the early nodes in the path, as these are the most crucial for good performance. This encoding of more information into the distance function is crucial to the success of the efficiency gradient.

I now speculate on some directions for future work in this area.

7.1 Structured Prediction

Obviously, there are myriad structured prediction domains that have herein not been explored, with machine translation [90] and natural language processing [19] being the ones that have driven the field thus far. Structured gradient boosting can be applied to many of the sub-tasks in these fields, and earlier results using the structured perceptron algorithm [50, 14] indicate that structured gradient boosting may find some success in these domains.

Another important thread of work in the structured prediction subfield is the exploration of the deep connections between the proposed algorithms. Structural

support vector machines, as stated before, already have some interesting theoretical guarantees [93], but these guarantees rely on exact inference during learning, as do the theoretical guarantees of some other methods [47, 91]. By contrast, some of the other methods [19, 67, 14] make no such requirement and are able to deliver other theoretical guarantees. Are there relationships between the two classes of methods and the problems that they are able to tackle? Are there other theoretical guarantees that cross the boundary between these two groups, in the same way VC theory [95] does for classical statistical learning?

It is assumed that exploration of the deep theoretical guarantees between methods will lead to empirical demonstrations of their various effectiveness and ineffectiveness on certain types of problems (or vice-versa). There is a lack of broad empirical comparison of methods in the structured prediction literature right now and this situation is starting to be remedied. Some recent workshop results[1] showed that violation of the exact inference requirement leads to poor performance of the SVM-struct algorithm. By contrast, violation of certain requirements of the structured perceptron algorithm [15] lead to *improved* convergence. More results like these will surely be of great value to the community in the future.

Finally, the feature function Ψ is assumed by all of the algorithms reviewed in this dissertation to be given. Induction of this function from training data is probably the single most important direction for future work is this area and one that so far has been virtually ignored.

[1] specifically, Thomas Finley's paper at the *ICML 2007 Workshop on Constrained Optimization and Structured Output Spaces* and Chris Mills-Price's paper at the *AAAI 2007 Workshop on Acquiring Planning Knowledge via Demonstration.*

7.2 Learning by Demonstration via Structured Prediction

This domain demonstrates that exact inference is not required for structured gradient boosting to be effective. In fact, this shows that learning can be effective using the simplest possible search procedure (random probing) in the inference routine of the algorithm. Dropping the requirement for exact (or even guaranteed approximate) inference in the inference routine may open up the structured prediction formalism model types other than the linear model and its kernelized variants.

In addition, the success of structured prediction in the learning by demonstration paradigm suggested possible success in related domains. In particular, I note that the experiments done in learning by demonstration treat a plan as a single, factored action with no temporal aspect. If we add the temporal aspect and consider a plan action by action, is it possible to use this result to improve results in the *reinforcement learning* paradigm? Some preliminary experiments have suggested that structured gradient boosting is able to learn many objectives faster than Q-learning [42]. I imagine an algorithm akin to approximate policy iteration [6] where the updates are done in the style of structured gradient boosting. Research investigating this idea is already underway.

7.3 Learning for Sequence Retrieval Accuracy

Obviously, one direction for future work is to apply this formalism to other sequence retrieval domains. The domains most amenable to these methods will likely exhibit one or both of the following characteristics: First, the elements of the candidate

sequences will have meaningful internal structure or relationships to one another. Second, the target set in the candidate domain will be fixed (or at least drawn from the same distribution) for both learning and performance, enabling discriminative approaches to encode the target set into the distance function. This approach will demonstrate a substantial advantage over traditional generative learning and the recently suggested SVM-align in situations where training data is sparse, or where the size of the alphabet from which the sequences are drawn is large or infinite.

Another direction for this work is to apply the methods here to structures other than sequences, such as trees or graphs. All that is required is an algorithm such as the Smith-Waterman algorithm, that takes two structures and "aligns" them, with respect to a set of costs, extracting the features of this alignment in the process. Recent work in the area of "tree edit distance" [8, 80] gives us a notion that this may be a useful area for exploration.

7.4 Learning for Sequence Retrieval Efficiency

The primary strength of the work in learning for efficiency is its novelty. While there have been a very few other modest attempts to tune a distance function to a particular metric access method [62], none of these has used machine learning to perform this tuning. Given these results at a first attempt, it is likely that future endeavors into this area will be fruitful.

Again, an obvious direction for future research is to apply the efficiency gradient to other domains. The "query-by-content" literature contains many such domains,

and these methods are applicable for any domain where a distance function is constructed. One possible candidate is the area of "content-based image retrieval", where a collection of images is searched using a query image. Recent studies [37] show that fractional L_p measures for content-based image retrieval are superior than traditional L_p measures. Because fractional L_p distances are non-metric, these learning methods may apply here.

A final future direction is to apply the efficiency gradient to other metric access methods. I noted previously that the success of the efficiency gradient lies in its ability to encode a *particular instance* of a metric access method into the distance function. Changes to the distance function will obviously invalidate parts of any metric access structure, but if the changes require that the structure be rebuilt entirely, all of the previous steps of the efficiency gradient are largely invalidated. Thus, the candidate metric access method must be *robust* to changes in the distance function. I have also tried this experiment using the recently proposed *cover trees* [7] as the metric access method. Unfortunately, cover trees are not robust to changes in the distance function, and so the efficiency gradient shows little effect when used with this type of structure. By contrast, *metric trees* [36] are even *more* invariant than vp-trees under changes to the distance function, and in preliminary tests they have shown at least equivalent performance.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML '04: Proceedings of the 21st International Conference on Machine Learning*, page 1, New York, NY, USA, 2004. ACM Press.

[2] D. Aha and D. Kibler. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecluar Biology*, 215:403–410, 1990.

[4] 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 Res.*, 25:3389–3402, 1997.

[5] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[6] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[7] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *ICML '06: Proceedings of the 23rd International Conference on Machine Mearning*, pages 97–104, 2006.

[8] Philip Bille. Tree edit distance, alignment distance and inclusion. Technical Report TR-2003-23, IT University of Copenhagen, March 2003.

[9] Paul Boersma. Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. *Proceedings of the Institute of Phonetic Sciences*, 17:97–110, 1993.

[10] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*, pages 43–49. Wadsworth and Brooks, Monterey, CA, 1984.

[11] Matthieu Carré, Pierrick Philippe, and Christophe Apélian. New query-by-humming music retrieval system conception and evaluation based on a query nature study. In *Proc. COST G-6 Conference on Digital Audio Effects*, Limerick, Ireland, 2001.

[12] Philip K. Chan, Wei Fan, Andreas L. Prodromidis, and Salvatore J. Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems*, 14(6):67–74, 1999.

[13] Eugene Charniak. *Statistical Language Learning*. MIT Press, Cambridge, MA, USA, 1994.

[14] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, 2002.

[15] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111, Morristown, NJ, USA, 2004.

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

[17] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March 2000.

[18] Roger B. Dannenberg, William P. Birmingham, George Tzanetakis, Colin Meek, Ning Hu, and Bryan Pardo. The musart testbed for query-by-humming evaluation. In *Proc. 4th International Symposium on Music Information Retrieval*, 2003.

[19] Hal Daumé III. *Practical Structured Learning Techniques for Natural Language Processing*. PhD thesis, University of Southern California, Los Angeles, CA, August 2006.

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

[21] Thomas G. Dietterich. Machine learning for sequential data: A review. *Structural, Syntactic, and Statistical Pattern Recognition; Lecture Notes in Computer Science*, 2396:15–30, 2002.

[22] Thomas G. Dietterich, Adam Ashenfelter, and Yaroslav Bulatov. Training conditional random fields via gradient tree boosting. In *International Conference on Machine Learning*, 2004.

[23] The Digital Tradition Folk Music Database. `http://www.mudcat.org`.

[24] W. Jay Dowling. Scale and contour: Two components of a theory of memory for melodies. *Psychological Review*, 85:341–354, 1978.

[25] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.

[26] Adriane Swalm Durey and Mark A. Clements. Melody spotting using hidden markov models. In *Proc. 2nd International Symposium on Music Information Retrieval*, Bloomington, IN, October 2001.

[27] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

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

[29] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–407, 2000.

[30] Asif Ghias, Jonathan Logan, David Chamberlin, and Brian C. Smith. Query by humming: Music information retrieval in an audio database. In *Proc. 3rd ACM Multimedia Conference*, pages 231–236, 1995.

[31] B. Gold and L. Rabiner. Parallel processing techniques for estimating pitch periods of speech in the time domain. *Journal of the Acoustical Society of America*, 46:442–448, 1969.

[32] R.P. Gorman and T.J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Journal of Neural Networks*, 1:75–89, 1988.

[33] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1998.

[34] David Heckerman. A Tutorial on Learning Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research, March 1995.

[35] David Heckermann. *Probabilistic similarity networks*. MIT Press, Cambridge, MA, USA, 1991.

[36] Gisli R. Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces (survey article). *ACM Transactions on Database Systems*, 28(4):517–580, 2003.

[37] Peter Howarth and Stefan M. Rüger. Fractional distance measures for content-based image retrieval. In *ECIR*, pages 447–456, 2005.

[38] Ning Hu, Roger Dannenburg, and Ann L. Lewis. A probabilistic model of melodic similarity. In *Proc. International Computer Music Conference*, 2002.

[39] Rong Jin, Rong Yan, Jian Zhang, and Alexander G. Hauptmann. A faster iterative scaling algorithm for conditional exponential model. In *ICML '03: Proceedings of the 20th International Conference on Machine Learning*, pages 282–289, 2003.

[40] Thorsten Joachims. Learning to align sequences, a maximum margin approach. Technical report, Cornell University, August 2003.

[41] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo, 1995. Morgan Kaufmann.

[42] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[43] Roni Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113(1-2):125–148, 1999.

[44] Youngmoo E. Kim, Wei Chai, Ricardo Garcia, and Barry Vercoe. Analysis of a countour-based representation for melody. In *Proc. 1st International Symposium on Music Information Retrieval*, October 2000.

[45] Naoko Kosugi and Yuichi Nishihara. A practical query-by-humming system for a large music database. In *Proc. 8th ACM Multimedia Conference*, 2000.

[46] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[47] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and lebeling sequence data. In *ICML '01: Proceedings of the 18th International Conference on Machine Learning*, 2001.

[48] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[49] D. J. Levitin. *Memory for Musical Attributes*, pages 214–215. MIT Press, Cambridge, MA, 1999.

[50] Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. An end-to-end discriminative approach to machine translation. In *International Conference on Computational Linguistics and Association for Computational Linguistics (COLING/ACL)*, 2006.

[51] A. T. Lindsay. Using contour as a mid-level representation of melody. Master's thesis, MIT Media Lab, 1996.

[52] Lei Lu, Hong You, and Hong-Jiang Zhang. A new approach to query by humming in music retrivial. In *Proc. IEEE International Conference on Multimedia and Expo*, August 2001.

[53] Ofer Matan, Henry S. Baird, Jane Bromley, Christopher J.C. Burges, John S. Denker, Lawrence D. Jackel, Yann Le Cun, Edwin P.D. Pednault, William D. Satterfield, Charles E. Stenard, and Timothy J. Thompson. Reading handwritten digits: A zip code recognition system. *Computer*, 25(7):59–63, 1992.

[54] Dominic Mazzoni and Roger B. Dannenberg. Melody matching directly from audio. In *Proc. 2nd Annual International Symposium on Music Information Retrieval*, 2001.

[55] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *ICML '00: Proceedings of*

the 17th International Conference on Machine Learning, Stanford, California, 2000.

[56] Rodger J. McNab, Lloyd A. Smith, Ian H. Witten, and Clare L. Henderson. Tune retrieval in the multimedia library. *Multimedia Tools and Applications*, 10(2/3):113–132, 2000.

[57] Colin Meek. *Modelling error in query-by-humming applications*. PhD thesis, The University of Michigan, 2004.

[58] Colin Meek and William Birmingham. Johnny can't sing. Technical Report CSE-TR-471-02, University of Michigan, 2002.

[59] Colin Meek and William Birmingham. Johnny can't sing: A comprehensive error model for sung music queries. In *Proc. 3rd International Symposium on Music Information Retrieval*, 2002.

[60] Colin Meek and William P. Birmingham. The dangers of parsimony in query-by-humming applications. In *Proc. 4th International Symposium on Music Information Retrieval*, 2003.

[61] Marcel Mongeau and David Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24:161–175, 1990.

[62] César Feijó Nadvorny and Carlos A. Heuser. Twisting the metric space to achieve better metric trees. In *XIX Simpósio Brasileiro de Bancos de Dados*, pages 178–190, 2004.

[63] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *ICML '00: Proceedings of the 17th International Conference on Machine Learning*, pages 663–670, 2000.

[64] Takuichi Nishimura, Hiroki Hashiguchi, Junko Takita, J. Xin Zhang, Masataka Goto, and Ryuichi Oka. Music signal spotting retrieval by a humming query using start frame feature dependent continuous dynamic programming. In *Proc. 2nd International Symposium on Music Information Retrieval*, Bloomington, IN, October 2001.

[65] Bryan Pardo and William Birmingham. Encoding timing information for musical query matching. In *Proc. 3rd International Symposium on Music Information Retrieval*, 2002.

[66] Bryan Pardo, William Birmingham, and Jonah Shifrin. Name that tune: A pilot study in finding a melody from a sung query. *Journal of the American Society for Information Science and Technology*, 55(4), 2004.

[67] Charles Parker, Alan Fern, and Prasad Tadepalli. Gradient boosting for sequence alignment. In *AAAI '06: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, 2006.

[68] Charles Parker, Alan Fern, and Prasad Tadepalli. Learning for efficient retrieval of structured data with noisy queries. In *ICML '07: The Twenty-forth International Conference on Machine Learning*, 2007.

[69] Steffen Pauws. Cubyhum: A fully operational query by humming system. In *Proc. 3rd International Symposium on Music Information Retrieval*, 2002.

[70] Vasin Punyakanok, Dan Roth, Wen tau Yih, and Dav Zimak. Learning and inference over constrained output. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI '05: Proceedings of the International Joint Conference on Aritificial Intelligence*, pages 1124–1129. Professional Book Center, 2005.

[71] Ross J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, January 1993.

[72] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, February 1989.

[73] M. Anand Raju, Bharat Sundaram, and Preeti Rao. TANSEN: A query-by-humming based music retrieval system. In *Proc. National Conference on Communications*, 2003.

[74] Preeti Rao and M. Anand Raju. Building a melody retrieval system. In *National Conference on Communications*, Bombay, India, 2002.

[75] Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. Maximum margin planning. In *ICML '06: Proceedings of the 23rd International Conference on Machine Learning*, pages 729–736, 2006.

[76] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386408, 1958.

[77] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter 20. Prentice Hall, second edition, 2003.

[78] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter 15. Prentice Hall, second edition, 2003.

[79] Robert E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[80] Dennis Shasha and Kaizhong Zhang. *Pattern Matching in String, Trees and Arrays*. Oxford University, 1997.

[81] Jonah Shifrin and William P. Birmingham. Effectiveness of HMM-based retrieval on large databases. In *Proc. 4th International Symposium on Music Information Retrieval*, 2003.

[82] Jonah Shifrin, Bryan Pardo, Colin Meek, and William Birmingham. HMM-based musical query retrieval. In *Proc. Joint Conference on Digital Libraries*, Portland, OR, 2002.

[83] Thomas R. Shultz, Denis Mareschal, and William C. Schmidt. Modeling cognitive development on balance scale phenomena. *Machine Learning*, 16(1-2):57–86, 1994.

[84] V. G. Sigillito, S. P. Wing, L. V. Hutton, and K. B. Baker. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262–266, 1989.

[85] Shriprakash Sinha. Leaf shape recognition via support vector machines with edit distance kernels. Master's thesis, Oregon State University, 2004.

[86] Tomáš Skopal. On fast non-metric similarity search by metric access methods. In *EDBT '06: Proceedings of the 10th International Conference on Extending Database Technology*, pages 718–736, 2006.

[87] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press, 1988.

[88] M. S. Smith and T. F. Waterman. Identification of common molecular subsequence. *Journal of Molecular Biology*, 147:195–197, 1981.

[89] Timo Sorsa and Jyri Huopaniemi. Melodic resolution in music retrieval. In *Proc. 2nd Annual International Symposium on Music Information Retrieval*, Bloomington, IN, October 2001.

[90] Ben Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford University, 2004.

[91] Ben Taskar, Carlos Guestrin, and Daphane Koller. Max margin markov networks. In *Advances in Neural Information Processing Systems*, 2004.

[92] Ioannis Tsochantaridis, Thoman Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML '04: Proceedings of the 21st International Conference on Machine Learning*, 2004.

[93] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[94] Alexandra Uitdenbogerd and Justin Zobel. Matching techniques for large music databases. In *Proc. 7th ACM Multimedia Conference*, 1999.

[95] Vladimir N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[96] A. J. Viterbi. Error bounds for convolutional codes and an asympotitcally optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–267, 1967.

[97] Jie Wei. Markov edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):311–321, March 2004.

[98] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools with Java Implementations*. Morgan Kaufmann, San Francisco, 2000.