Gradient Boosting for Sequence Alignment

Charles Parker
Oregon State University
Kelley Engineering Center
Corvallis, OR 97333
parker@cs.orst.edu

Alan Fern
Oregon State University
Kelley Engineering Center
Corvallis, OR 97333
parker@cs.orst.edu

Prasad Tadepalli
Oregon State University
Kelley Engineering Center
Corvallis, OR 97333
tadepall@cs.orst.edu

Abstract

Sequence alignment is a common subtask in many applications such as genetic matching and music information retrieval. Crucial to the performance of any sequence alignment algorithm is an accurate model of the reward of transforming one sequence into another. Using this model, we can find the optimal alignment of two sequences or perform query-based selection from a database of target sequences with a dynamic programming approach. In this paper, we describe a new algorithm to learn the reward models from positive and negative examples of matching sequences. We develop a gradient boosting approach that reduces sequence learning to a series of standard function approximation problems that can be solved by any function approximator. A key advantage of this approach is that it is able to induce complex features using function approximation rather than relying on the user to predefine such features. Our experiments on synthetic data and a fairly complex real-world music retrieval domain demonstrate that our approach can achieve better accuracy and faster learning compared to a state-of-the-art structured SVM approach.

Introduction

The problem of sequential alignment and selection is not new to the literature. Over the last 40+ years, it has been shown that solutions to this problem are useful in a number of ways - from common applications like protein similarity detection (Krogh *et al.* 1994) and speech recognition (Rabiner 1989) to more exotic applications like plant identification (Sinha 2004) and music information retrieval (Dannenberg *et al.* 2003).

Most if not all of the solutions developed focus around a set of *reward functions* or *edit distance functions* that describe the proximity of one sequence to another. Such functions can be put to a variety of uses, including retrieving similar sequences from databases, clustering sequences into hierarchical classes, and evaluating the similarity between two sequences. Unfortunately, effective reward functions can be highly complex and dependent on the domain. Rather than having to design them by hand for every new domain, we would like to be able to learn approximate versions of the functions, given some alignments for training purposes. Since training data is usually expensive to collect, we want to be able to learn as efficiently as possible.

Most prior work has used generative modeling to learn these functions, and typically requires the sequence elements to be characters from a finite alphabet. Thus, offering no mechanism for generalizing the learning to characters not present in the training data (Meek 2004). More recently, discriminative learning methods, based on support vector machines (SVM), have been developed for this problem (Tsochantaridis *et al.* 2004). This has lead to improved accuracy, in some domains, at a reward of increased training time. Still, however, the existing SVM-based system does not address the issue of generalization over sequence elements.

In this paper, we describe a new learning approach for aligning sequences of rich objects with informative internal structure. Our technique is discriminative like the SVM-based approach but can be trained in a fraction of the time. Our approach also facilitates a type of automated feature construction, helping to avoid the need for substantial feature engineering. The main idea is to convert Friedman's loss function from "LogitBoost" (Friedman, Hastie, & Tibshirani 2000) to a loss function meaningful for sequence learning, then to use a function approximator to estimate the gradient function at points unseen in the training data. We then demonstrate experiments on synthetic data and real-world data from the music retrieval domain that show that our approach can perform better than the state-of-the-art SVM-based technique at a fraction of the training time, as well as outperforming the traditional generative modeling approach.

Sequence Alignment

A *sequence-matching problem* is a tuple $(\mathcal{T}, D, Q, \mathcal{P})$, where $\mathcal{T}$ is a set of finite *target sequences* $\{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_k\}$ over the *target domain* D, i.e. $\mathbf{t}_i \in D^*$. Q is the *query domain*, letting $\mathcal{Q} = Q^*$ be the set of all finite *query sequences* constructed from Q. Finally, $\mathcal{P}$ is a probability distribution over $\mathcal{T} \times \mathcal{Q}$. Intuitively, $\mathcal{P}$ is a distribution over possible queries paired with the correct target sequences. The objective is to find a *matching function* f such that for target-query pairs $(\mathbf{t}, \mathbf{q})$ drawn from $\mathcal{P}$ there is a high probability that $f(\mathbf{q}, \mathcal{T}) = \text{argmax}_{\mathbf{t} \in \mathcal{T}} \mathcal{P}(\mathbf{t}, \mathbf{q})$.

To give a more concrete example, consider the speech recognition domain. The target set is a dictionary of words, represented as sequences of *phonemes*. A query is a sequence of phones (a word) spoken by a user, and the task is to select the word spoken by the user from the dictionary. Typically, this is done by designing a *scoring function* $F : \mathcal{T} \times \mathcal{Q} \mapsto \Re$ which is to be maximized at the correct output, so that:

$$f(\mathbf{q}, \mathcal{T}) = \operatorname*{argmax}_{\mathbf{t} \in \mathcal{T}} F(\mathbf{t}, \mathbf{q})$$

Typically the scoring function F is defined using the notion of sequence alignment. That is, $F(\mathbf{t}, \mathbf{q})$ returns the reward for the highest reward series of edit operations (such as match, insert, and delete) that transforms the target $\mathbf{t}$ into the query sequence $\mathbf{q}$. Each edit operation is associated with a reward function. In this paper we consider the operations match, insert, and delete with associated reward functions: $r_m(a, b)$ giving the reward for replacing character a in the target with b, $r_i(a)$ giving the reward for inserting a, and $r_d(a)$ giving the reward for deleting a. These reward functions serve as our parameterization of F. Figure 1 shows an alignment between the target sequence `CABIN` and the query sequence `DRAIN`[1], indicating the series of edit operations in the top row. The reward for the alignment is the sum of the individual edit operations.

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

Figure 1: A Lexicographical Example of Sequential Alignment.

To find the highest reward alignment of two sequences we use the Smith-Waterman algorithm (1981). Let $\text{align}(i, j, \mathbf{t}, \mathbf{q})$ be the reward of the optimal alignment between the postfix of $\mathbf{t}$ starting at position i and the postfix of $\mathbf{q}$ starting at j. This function has the following decomposition,

$$\text{align}(i, j, \mathbf{t}, \mathbf{q}) = \max \begin{cases} r_m(t_i, q_j) + \text{align}(i+1, j+1, \mathbf{t}, \mathbf{q}) \\ r_i(q_j) + \text{align}(i, j+1, \mathbf{t}, \mathbf{q}) \\ r_d(t_i) + \text{align}(i+1, j, \mathbf{t}, \mathbf{q}) \end{cases} \quad (1)$$

which yields a straightforward dynamic programming procedure for computing the optimal alignment. The procedure completes in time proportional to the product of the sequence lengths. One can define versions of the algorithm that ignore prefixes or suffixes in the query as desired. See (Meek 2004) for further discussion.

Learning Reward Functions

We have seen how to apply sequence alignment to the problem of sequence matching. However, sequence alignment requires that we provide a reward function, and high-performance reward functions tend to be highly complex and domain specific. Thus, significant effort has been invested in techniques for automatically learning reward functions from examples. In this learning problem, the training examples are target and query sequences, properly aligned, and assigned a correct score. Our goal is to learn the functions r_m, r_i, and r_d mentioned above. Here we explore two primary techniques for learning these functions, and discuss some shortcomings that our work addresses.

Generative Approach

The generative approach to learning reward functions assumes a probabilistic model for generating queries from target sequences (1998). 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. We take the reward for an alignment to be the logarithm of the probability of the alignment.

Under this model our reward functions become log probabilities of the edit operation probabilities:

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

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.

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 $\mathbf{q}$. Note that $\mathbf{a}$ is simply a sequence of the edit operations or events. 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 r_m 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 r_i and r_d. 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.

There are at least two problems with this method. First, the generative training of the model does not take advantage of the discriminatory nature of the underlying task, and

[1]Thanks to (Meek 2004) for the example.

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 (Lafferty, McCallum, & Pereira 2001; Taskar, Guestrin, & Koller 2004). Second, it seems that there is no way to extend the learning process so that the reward behavior of one character 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.

We 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 in Section 4 can be viewed as a way to avoid the need to explicitly make such choices.

SVM-align

Recent work on sequence-alignment learning addresses the first of the above problems by developing a more discriminative approach (Joachims 2003; Tsochantaridis *et al.* 2004). Here, we are again given a series of alignments of query and correct target as training data. However, we are also given a number of *incorrect* or *decoy* targets. That is, our training set $\mathcal{S}$ contains tuples $(\mathbf{t}, \mathbf{q}, \mathbf{a}, \mathcal{N})$, where $\mathbf{a}$ is the best alignment between the query $\mathbf{q}$ and correct target $\mathbf{t}$ and $\mathcal{N} \subseteq \mathcal{T}$ is a set of incorrect target sequences. The learning problem is then formulated as a constrained optimization problem.

More formally, let $\Psi(\mathbf{t}, \mathbf{q}, \mathbf{a})$ be a vector of counts of each possible edit event in the alignment (e.g. the number of times 'a' was matched with 'b'). If we let the weight vector $\mathbf{w}$ represent the rewards for each of the possible events, then the total reward of an alignment is given by $\langle w, \Psi(\mathbf{t}, \mathbf{q}, \mathbf{a})\rangle$ and the corresponding scoring function is given by,

$$F(\mathbf{t}, \mathbf{q}) = \max_{\mathbf{a}} \langle w, \Psi(\mathbf{t}, \mathbf{q}, \mathbf{a})\rangle$$

Our goal, is to learn weights $\mathbf{w}$ such that for query-target pairs $(\mathbf{q}, \mathbf{t})$ drawn from our problem distribution the score of the correct target $\mathbf{t}$ 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 $(\mathbf{t}, \mathbf{q}, \mathbf{a}, \mathcal{N})$ we have,

$$\forall \mathbf{t}' \in \mathcal{N},$$
$$\langle \mathbf{w}, \Psi(\mathbf{t}, \mathbf{q}, \mathbf{a})\rangle - \max_{a'} \langle \mathbf{w}, \Psi(\mathbf{t}', \mathbf{q}, a')\rangle > 0 \quad (2)$$

There may be many $\mathbf{w}$ that satisfy these constraints, and as customary for SVMs (Vapnik 1998), the SVM-align approach selects the $\mathbf{w}$ that maximizes a suitable notion of margin. This is non-trivial constrained optimization problem since the number of constraints in Equation 2 is on the order of the number of possible sequence alignments, which is exponential in the length of the sequences. However, (Joachims 2003) shows that near-optimal solutions can be found in polynomial time using an incremental constraint generation approach. While polynomial time, this approach to sequence-alignment learning is typically orders of magnitude slower than generative approaches.

While this technique provides a powerful discriminative approach to sequence-alignment learning, it still does not address the latter 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.

Though an implementation is not available, we note that 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.

Application of Gradient Boosting

In our motivating application of music retrieval, the sequence elements are vectors of real-valued features extracted via standard music processing. Thus, a straightforward application of the above approaches requires flattening the vector space into single character elements, which as discussed above can be undesirable. In addition, extending the above approaches to directly utilize the element structure requires significant insight into the domain. That is, good reward functions for this application appear to be non-linear in the primitive features, meaning that one must hand-engineer appropriate non-linear features to be used by SVM-align, or carefully choose the probabilistic model used by a generative approach. These difficulties motivated us to develop a new approach that can more powerfully utilize a given representation.

Our approach is inspired by the gradient boosting framework of (Friedman, Hastie, & Tibshirani 2000), which has been previously applied to solve classification and regression problems and more recently to train conditional random fields (Dietterich, Ashenfelter, & Bulatov 2004). We begin by defining our loss function which is based on the margin-based loss function used by the "LogitBoost" procedure (Friedman, Hastie, & Tibshirani 2000) for binary classification:

$$\log(1 + \exp(-2y_i F(x_i))) \quad (3)$$

where the *margin* in this loss function is $2y_i F(x_i)$. This margin only makes sense for binary classification, as in Friedman's paper, but we can make it meaningful in our application by redefining the margin. A reasonable definition for the margin in this application is the difference, for a given

query $\mathbf{q}_i$, between the correct target and the highest scoring incorrect target. Suppose that $\mathbf{t}_i$ is the correct target for query sequence $\mathbf{q}_i$ and $\hat{\mathbf{t}}_i$ is the highest scoring incorrect target for this query under alignment $\hat{\mathbf{a}}_i$. If $F(\mathbf{x}_i, \mathbf{y}_i, \mathbf{a}_i)$ is the reward for aligning sequence $\mathbf{x}$ to sequence $\mathbf{y}$ with the series of alignment operations $\mathbf{a}_i$, then we define the margin for this example to be:

$$F(\mathbf{t}_i, \mathbf{q}_i, \mathbf{a}_i) - F(\hat{\mathbf{t}}_i, \mathbf{q}_i, \hat{\mathbf{a}}_i) \quad (4)$$

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

$$\log(1 + \exp(F(\hat{\mathbf{t}}_i, \mathbf{q}_i, \hat{\mathbf{a}}_i) - F(\mathbf{t}_i, \mathbf{q}_i, \mathbf{a}_i)) \quad (5)$$

Notice that the loss function monotonically decreases with increasing margin as desired.

We can express our scoring function, F as a sum of the rewards for the various events in the alignment. To do this, let us define $r(a, b)$ to represent a combination of the three reward functions discussed in previous sections (where a and b can be a null character in the case of insertions and deletions, respectively). Also define $f(a, b, \mathbf{x}_i, \mathbf{y}_i, \mathbf{a}_i)$ to be the number of times that character a is replaced by b in the alignment $\mathbf{a}_i$ of sequence $\mathbf{x}_i$ with sequence $\mathbf{y}_i$. Thus we can express the scoring function as:

$$F(\mathbf{t}_i, \mathbf{q}_i, \mathbf{a}_i) = \sum_a \sum_b r(a, b)[f(a, b, \mathbf{t}_i, \mathbf{q}_i, \mathbf{a}_i)] \quad (6)$$

With this formulation, our learning process, which is a direct application of functional gradient descent, is to iteratively learn the function $r(a, b)$ in order to minimize the loss function. More specifically let r_k be the reward function after k iterations. Initially we set r_0 to be a human provided function, perhaps based on prior knowledge or uninformative. Given r_k we compute r_{k+1} by approximating the functional gradient δ_{k+1} of the loss function with respect to r_k and then setting $r_{k+1} = r_k - \delta_{k+1}$. This tends to move r_{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 we now describe.

To approximate the functional gradient δ_{k+1} we will estimate 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 $\{\langle (a, b), \delta_{k+1}(a, b)\rangle\}$ which can be passed to a function approximator yielding an approximation δ_{k+1} that generalizes across all possible pairs (a, b). The key step in creating the training set then is to compute the functional gradient of the loss function for a given pair, which we describe below.

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

$$\Delta f_i(a, b) = f(a, b, \mathbf{t}_i, \mathbf{q}_i, \mathbf{a}_i) - f(a, b, \hat{\mathbf{t}}_i, \mathbf{q}_i, \hat{\mathbf{a}}_i)$$

With this definition the cumulative loss over the training set is given by:

$$L = \sum_i \log[1 + \exp(-\sum_a \sum_b r(a, b)[\Delta f_i(a, b)])]$$

and the function gradient at pair (a, b) is derived as follows:

$$\begin{aligned}\delta_{k+1}(a, b) &= \frac{\partial L}{\partial r_k(a, b)} \\ &= -\sum_i \frac{\exp(-\sum_{a'} \sum_{b'} r_k(a', b')\Delta f_i(a', b'))\Delta f_i(a, b)}{1 + \exp(-\sum_{a'} \sum_{b'} r_k(a', b')\Delta f_i(a', b'))} \\ &= -\sum_i \frac{\Delta f_i(a, b)}{1 + \exp(\sum_{a'} \sum_{b'} r_k(a', b')\Delta f_i(a', b'))}\end{aligned}$$

Intuitively, our gradient function attempts to increase the rewards of events occurring in positive alignments and decrease the events occurring in negative alignments at each iteration. In this way, we have maintained the discriminative learning aspect of SVM-align. However, our approach is much more computationally efficient and able to generalize over the character space. We accomplish both of these goals by training regression trees to approximate each functional gradient δ_k. Of course, any other function approximator could be applied to the training data. This bodes well for our method given the amount of work on function approximation that can be found in the literature.

In addition to this computational advantage we now have a representational advantage: There are now no restrictions on the way in which our sequence elements must be represented. We are equally as comfortable with a finite alphabet as we are with real numbers, or vectors of real numbers as our sequence elements, so long as we have an appropriate function approximator for the data type. Neither are we limited to linear cost functions as we can simply use non-linear approximators to represent complex relationships among the primitive features of the sequence elements.

Experiments

In the following experiments, we run over training data sets of different sizes with a fixed target set size. For each training set size, we learn 100 models for each learning process with randomly selected training data, to minimize the effects of randomness. For both domains, *Error* is the fraction of the time the correct target is not selected from the target set based on a given query. For SVM-align, we use the default options. Regression trees are trained to minimize squared-error

Synthetic Domain

Our first set of training data is generated from a synthetic domain with structured elements designed to show the benefit of exploiting structure in domains where it is present. 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. 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).

Query-by-humming Domain

In the query-by-humming domain (Meek 2004; Dannenberg *et al.* 2003), we are given as a query an audio file of a person singing or playing a song. We have a target database of sequences of notes. Our job is to match the audio file to the correct sequence of notes.

The process of "transcribing" a series of notes from a sung query is well-documented elsewhere (Meek 2004) and not of particular interest here. When transcription is complete, the events in the query sequence will be real and vector-valued, containing a component for both the *average pitch* and the *duration* of an event, so a query song $\mathbf{s}$ is represented by a series of tuples:

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

Furthermore, the pitch and duration of the starting event is immaterial so long as the proper *relative* pitches and durations are maintained. Thus, we instead represent the song 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 $s_i^r = \frac{s_{i+1}^d}{s_i^d}$.

We discretize and flatten these real-valued quantities to a finite alphabet for the purposes of training in the SVM-align and Bayesian formalisms. For training with our method, the data remains real and vector-valued. If we use the levels of granularity suggested in the literature (Carré, Philippe, & Apélian 2001; Pardo & Birmingham 2002), we have 27 levels for pitch difference and 4 for duration ratio, giving $27 \times 4 = 108$ characters in the alphabet.

Our query set consists of 587 queries collected by the first author from college and church choirs. Twelve target songs and 50 different singers were used to generate these queries. To compose our training and test data, we first split both targets and queries into training and test sets, then we draw randomly from the these sets to compose the training and test data. For the target sets we draw from the *Digital Tradition* database of folk melodies in monophonic midi format. To generate alignments for training, we use the state of the art "note-interval" reward model described in (Dannenberg *et al.* 2003).

Results

Synthetic Domain In the synthetic domain, our method has an advantage over both the traditional Bayesian method and SVM-align, as shown in Figure 2. Both SVM-align and our method are able to learn a reasonable model with less training data than the Bayesian method, but with about 15 training sequences, our method is able to begin exploiting the structure of the domain that the other two methods ignore.

In Figure 3, we see that our method also has an advantage in computation time over SVM-align even at small test set sizes, and the gap grows dramatically even as we approach a training set as small as 40 sequences. We also note that the code for our method is running as interpreted Java inside of a development environment, whereas the SVM-align code is compiled C. The difference, therefore, is actually much larger than it appears.

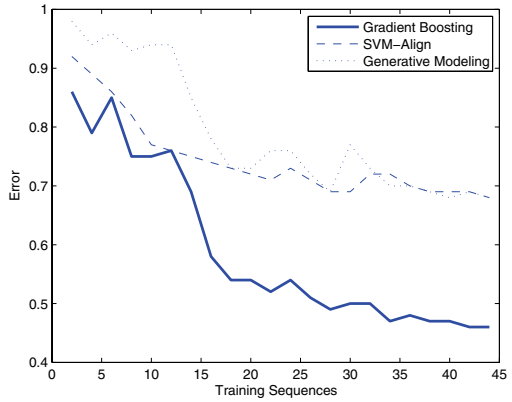

Figure 2: Learning curves for all models in the synthetic domain. Training sequences are an average of 5 elements in length.

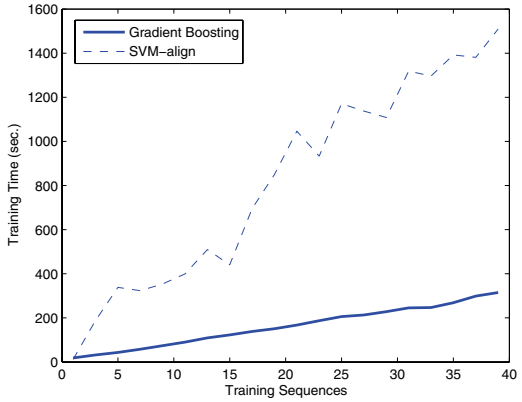

Figure 3: Running time for gradient boosting vs. SVM align in the synthetic domain. Training sequences are an average of 30 elements in length.

Query-by-humming Domain In the query-by-humming domain, we see that the SVM-align moves quickly to a reasonable error rate, but at around 60 training sequences, SVM-align is overtaken by gradient boosting. Eventually, with about twice as much training data, SVM-align manages to catch up to gradient boosting. This provides evidence that gradient boosting is able to generalize to unseen data with greater ease than SVM-align. We note that both methods

vastly outperform the traditional Bayesian learning methods, in keeping with the results found in (Tsochantaridis *et al.* 2004).

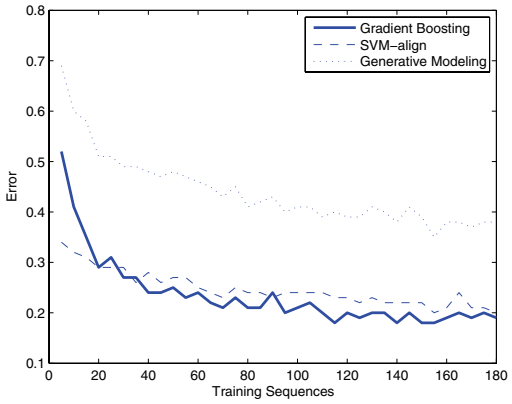

Figure 4: Learning curves for the models in the query-by-humming domain. Target set size is fixed at 50.

Conclusions and Future Work

We have shown a new algorithm for sequence alignment based on gradient tree boosting and LogitBoost. We have shown that this algorithm is able to perform as well or better than two competing techniques in both a synthetic and a real-world domain in a fraction of the training time. We have also seen that it is highly flexible in two ways: That it can use any regression algorithm and that the features given to the algorithm need not be symbols from a single-character alphabet.

A further comparison of gradient boosting to SVM-align would involve incorporating kernels. However, conversations with the developers of SVM-align showed that it is not obvious how to incorporate kernels into the method. The only obvious way to tailor the method to a particular application is to engage in the time-consuming and domain-specific feature construction discussed earlier. If kernels are incorporated into SVM-align at a later date, a comparison will be warranted.

Also, while we have seen good performance with the music and synthetic data, one can imagine that there are other domains where the reward relationship between edit operations can be exploited further. Another thrust of future work is to investigate other domains where gradient boosting can be useful.

Another interesting consequence of our method's construction is a unique ability to incorporate prior knowledge. Our initial reward function r_0 was uninformative for our experiments here, but experiments using the state-of-the-art reward function in (Dannenberg *et al.* 2003) as the initial reward function show that gradient boosting can improve the accuracy of the state-of-the-art function by about 10%.

Finally, one of the keys to SVM-align is that the formalism is extensible to all types of structured data. Future work may attempt to apply the gradient boosting method to structured data such as trees or graphs in which the elements of the structures have complex structure themselves.

References

Carré, M.; Philippe, P.; and Apélian, C. 2001. 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*.

Dannenberg, R. B.; Birmingham, W. P.; Tzanetakis, G.; Meek, C.; Hu, N.; and Pardo, B. 2003. The musart testbed for query-by-humming evaluation. In *Proc. 4th International Symposium on Music Information Retrieval*.

Dietterich, T. G.; Ashenfelter, A.; and Bulatov, Y. 2004. Training conditional random fields via gradient tree boosting. In *International Conference on Machine Learning*.

Durbin, R.; Eddy, S.; Krogh, A.; and Mitchison, G. 1998. *Biological Sequence Analysis*. Cambridge University Press.

Friedman, J.; Hastie, T.; and Tibshirani, R. 2000. Additive logistic regression: a statistical view of boosting. *Annals of Statistics* 28(2):337–407.

Joachims, T. 2003. Learning to align sequences, a maximum margin approach. Technical report, Cornell University.

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

Lafferty, J.; McCallum, A.; and Pereira, F. 2001. Conditional random fields: Probabilistic models for segmenting and lebeling sequence data. In *ICML*.

Meek, C. 2004. *Modelling error in query-by-humming applications*. Ph.D. Dissertation, The University of Michigan.

Pardo, B., and Birmingham, W. 2002. Encoding timing information for musical query matching. In *Proc. 3rd International Symposium on Music Information Retrieval*.

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

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

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

Taskar, B.; Guestrin, C.; and Koller, D. 2004. Max margin markov networks. In *NIPS*.

Tsochantaridis, I.; Hofmann, T.; Joachims, T.; and Altun, Y. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proc. 21st International Conference on Machine Learning*.

Vapnik, V. N. 1998. *Statistical Learning Theory*. John Wiley & Sons.