Learning for Efficient Retrieval of Structured Data with Noisy Queries

Charles Parker PARKER@EECS.OREGONSTATE.EDU
Alan Fern AFERN@EECS.OREGONSTATE.EDU
Prasad Tadepalli TADEPALL@EECS.OREGONSTATE.EDU
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, 97331-5501

Abstract

Increasingly large collections of structured data necessitate the development of efficient, noise-tolerant retrieval tools. In this work, we consider this issue and describe an approach to learn a similarity function that is not only accurate, but that also increases the effectiveness of retrieval data structures. We present an algorithm that uses functional gradient boosting to maximize both retrieval accuracy and the retrieval efficiency of vantage point trees. We demonstrate the effectiveness of our approach on two datasets, including a moderately sized real-world dataset of folk music.

1. Introduction

The enormous storage capacity of modern computer systems has led to huge collections of structured data including images, audio clips, and DNA sequences. This in turn necessitates the development of effective tools for efficient and noise-tolerant retrieval of such structured data through *natural* queries, i.e., using data of the type that is in the dataset. The so-called "query-by-content" (Faloutsos et al., 1994; Ghias et al., 1995) literature aims at achieving exactly this: Giving the ability to query collections of music using music, images using images, and gene sequences using other gene sequences.

Since the queries are in general noisy versions of the targets, the problem of retrieving the closest match to the query can be formalized by defining a *distance function* d over the domain of possible structures $\mathcal{D}$ so that $d : \mathcal{D} \times \mathcal{D} \mapsto \Re$ represents the dissimilarity between the target and the query. Retrieving the nearest neighbor of a query structure $\mathbf{q}$ in a set of targets $\mathcal{T}$ under this definition, then, is simply finding $\text{argmin}_{\mathbf{t} \in \mathcal{T}}\, d(\mathbf{t}, \mathbf{q})$.

In this paper, we consider the problem of learning the distance function from a training dataset $\mathcal{S}$ over a target set $\mathcal{T}$, with tuples of the form $(\mathbf{t}, \mathbf{q})$ where $\mathbf{t} \in \mathcal{T}$ and $\mathbf{q}$ is a query for which the correct answer is $\mathbf{t}$.

Previous attempts to learn distance functions (Pardo et al., 2004; Tsochantaridis et al., 2004; Parker et al., 2006) in these domains have had a great deal of success, achieving high degrees of accuracy with modest training data. However, the distance functions are often highly complex, making the computation of $\text{argmin}_{\mathbf{t} \in \mathcal{T}}\, d(q, \mathbf{t})$ through sequential search very expensive for large $|\mathcal{T}|$. A group of data structures, known collectively as *metric access methods* or MAMs, help avoid the sequential search. The general approach is to use a tree that partitions the structure space based on distance. Given this tree and a query, groups of targets at long distances from the query can be eliminated from the search with a single distance computation. Cover trees (Beygelzimer et al., 2006), k-d trees, m-trees, and vp-trees (Hjaltason & Samet, 2003) are all examples of this approach.

There are some problems to be overcome in using these methods: First, to use an MAM for search, d must be a *metric distance function*. Unfortunately, in many practical cases, the learned distance function is not a metric (Skopal, 2006). Second, there are few guarantees that MAMs will provide increased search speed even if d is metric (Weber et al., 1998).

Recent work (Skopal, 2006) has addressed the former concern. We here turn our attention to the latter. In Section 2, we introduce the problem of learning distance functions for sequence matching and outline a

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

gradient boosting algorithm for it. In Section 3, we present an extension of this algorithm that maintains high accuracy and increases retrieval efficiency. This is accomplished by first building a vantage-point tree (vp-tree) over the given data. We then derive a definition of loss and margin based on the constructed tree and maximize this margin using gradient descent boosting. Section 4 shows experimental results in two sequence matching domains including a real-world collection of folk songs. Section 5 discusses the results and speculates on the possible extensions.

2. Learning to Match Sequences

In this section, we introduce the sequence matching problems and describe a gradient boosting approach to learning an edit distance function, which is central to sequence matching.

2.1. Sequence Matching Problems

A variety of sequence matching problems are already well-documented in the literature (Pardo et al., 2004; Krogh et al., 1994). In a generic version of the problem, we are given a set of target sequences $\mathcal{T}$. Associated with this target set is a set of query sequences $\mathcal{Q}$, each having a correctly matching target in $\mathcal{T}$. The goal is to design a function that maps all queries onto their correct targets. In the speech recognition domain, these sequences are composed of phonemes. Targets are the dictionary sequences and queries are sequences spoken by a user.

Typically, the mapping from queries to targets is done by designing a distance function $d : \mathcal{T} \times \mathcal{Q} \mapsto \Re$ which is to be minimized at the correct target. If $\mathbf{t}_i \in \mathcal{T}$ is target mapping correctly to a query $\mathbf{q}_i$, then our distance function should satisfy $\mathbf{t}_i = \text{argmin}_{\mathbf{t} \in \mathcal{T}}\, d(\mathbf{t}, \mathbf{q}_i)$ for all $(\mathbf{t}_i \in \mathcal{T}, \mathbf{q}_i \in \mathcal{Q})$.

The distance function d is often defined using the notion of sequence alignment. That is, $d(\mathbf{t}, \mathbf{q})$ returns a distance that is the sum of the lowest cost series of edit operations (**m**atch, **r**eplace, **d**elete, and **i**nsert) that transforms the target $\mathbf{t}$ into the query sequence $\mathbf{q}$. Each edit operation is associated with a costs c such that $c(a, b)$ gives the cost for matching character a in the target with b, $c(a, -)$ gives the cost for inserting a, and $c(-, a)$ gives the cost for deleting a. These cost functions serve as our parameterization of d. Figure 1 shows an alignment between the target sequence `BRAIN` and the query sequence `BLANE`, indicating the series of edit operations in the top row.

The lowest cost alignment of two sequences can be found efficiently using the Smith-Waterman algorithm

m	**r**	**m**	**d**	**m**	**i**
B	R	A	I	N	-
B	L	A	-	N	E

Figure 1. An Example of Sequence Alignment.

(Smith & Waterman, 1981). Let $\text{align}(i, j, \mathbf{t}, \mathbf{q})$ be the cost of the optimal alignment between the postfix of $\mathbf{t}$ starting at position i with character t_i and the postfix of $\mathbf{q}$ starting at j with character q_j. This function has the following decomposition,

$$\text{align}(i, j, \mathbf{t}, \mathbf{q}) = \min \begin{cases} c(t_i, q_j) + \text{align}(i+1, j+1, \mathbf{t}, \mathbf{q}) \\ c(-, q_j) + \text{align}(i, j+1, \mathbf{t}, \mathbf{q}) \\ c(t_i, -) + \text{align}(i+1, j, \mathbf{t}, \mathbf{q}) \end{cases} \quad (1)$$

Using dynamic programming, this procedure completes in time proportional to the product of the sequence lengths. One can also define versions of the algorithm that ignore prefixes or suffixes in the query or target. See Meek (2004) for further discussion.

Although the cost function can occasionally be constructed by hand, it is often useful to be able to learn cost functions from a training set with examples of queries aligned with their proper targets. Historically, generative approaches have been favored for this task (Krogh et al., 1994). In recent years, however, several discriminative approaches (Tsochantaridis et al., 2004; Joachims, 2003) have been presented that appear to be more effective in some sequence matching domains. For this work, we use one of these recently suggested methods, discussed in the following section.

2.2. Gradient Boosting

In previous work, we advocated the use of *gradient boosting* as a method for learning edit distance functions (Parker et al., 2006). We will briefly revisit the derivation, as it provides the basis for our new derivation in Section 3.3.

The basic idea is to iteratively learn the distance function by calculating the gradient at each replacement pair (a, b). The gradient is designed to push the cost of the replacement $c(a, b)$ in the direction that moves the training queries closer to the *correct* targets and further from the *closest incorrect targets*. In spirit then, the algorithm is a margin-maximization approach.

We first define the margin for each training example $\mathbf{q}_i$. Define $\mathbf{t}_i$ to be the correct target given the query $\mathbf{q}_i$, and define $\hat{\mathbf{t}_i}$ to be the closest incorrect target given the current set of replacement costs c_k. Our structure

margin s_i at $\mathbf{q}_i$ is defined as how much closer $\mathbf{q}_i$ is to the correct target than to the closest incorrect target under d:

$$s_i = d(\hat{\mathbf{t}_i}, \mathbf{q}) - d(\mathbf{t}_i, \mathbf{q}) \quad (2)$$

Now define $f(a, b, \mathbf{x}, \mathbf{y}, \mathbf{a})$ to be the number of times character a is replaced with character b in the optimal alignment $\mathbf{a}$ of sequences $\mathbf{x}$ and $\mathbf{y}$. If $\mathbf{a}_i$ is the optimal alignment of $\mathbf{q}_i$ and $\mathbf{t}_i$ then the distance function decomposes nicely as

$$d(\mathbf{q}_i, \mathbf{t}_i) = \sum_a \sum_b c_k(a, b) f(a, b, \mathbf{t}_i, \mathbf{q}_i, \mathbf{a}_i) \quad (3)$$

with a similar expression for $\hat{\mathbf{t}_i}$ and its optimal alignment $\hat{\mathbf{a}_i}$. Defining the shorthand

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

then substituting into the original expression of margin gives us

$$s_i = -\sum_a \sum_b c_k(a, b) \Delta f_i(a, b) \quad (4)$$

To form a loss function from this margin, a technique is borrowed from *LogitBoost* (Friedman et al., 2000), where the loss function is given as $\log(1 + \exp(s_i))$ Replacing s_i with the margin in Equation 4, we have the loss defined for a single training query $\mathbf{q}_i$. If we sum all of these losses, we have the cumulative loss L over the entire training set.

$$L = \sum_i \log[1 + \exp(-\sum_a \sum_b c(a, b) \Delta f_i(a, b))] \quad (5)$$

To minimize this loss over the training set, we compute the gradient, δ_{k+1}, at each pair:

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

The gradient step δ_{k+1} is then added to the current set of replacement costs c_k, thus modifying the distance function, and the process is repeated. As the iterations progress, we expect that the distance function will become increasingly more accurate on the training queries In practice, Parker et al. (2006) gives some convincing empirical demonstrations.

In the next section, we first explain metric access methods and their use. We then attempt to use gradient boosting to learn a distance function that retrieves with high efficiency under a given metric access data structure while maintaining high accuracy.

3. Optimizing for Efficiency

We have seen in the previous section how 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 reasonably large target sets (Dannenberg et al., 2003). Moreover, the matching of the query to every possible target in the target set seems intuitively unnecessary.

Metric access methods attempt to remedy this situation. 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. Although the details of this structure vary from method to method, these trees all require that the distance function be a *metric*. This means it must satisfy four criteria for all elements in the domain, namely, positivity, identity, symmetry, and the triangular inequality.

The first three can be satisfied with relative ease in the sequence alignment domains. Positivity can be enforced by taking negative 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. Identity can be enforced just 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-dependant ways. The triangular inequality is far more difficult. Recent work, however, aims at modifying a distance function so that it satisfies the triangular inequality with high probability. It is this work that we review next.

3.1. 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 much greater than the distance 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 have maintained the retrieval accuracy and transformed d into a metric.

The insight of Skopal (2006) is that the simple ap-

plication 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 to where we started. We would like then, to have a function g with concavity sufficient to repair all non-triangular triples, but no more.

For example, Skopal (2006) proposes the function $g(x) = x^{\frac{1}{1+w}}$. If we apply this to d, we have a new distance metric.

$$d_g(x, y) = g(d(x, y)) = d(x, y)^{\frac{1}{1+w}} \tag{6}$$

As $w \to \infty$, the function reaches maximum concavity 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 MAMs. To check how well a given d_g satisfies the triangle inequality, we can sample triples from $\mathcal{T}$. Our experiments in Section 4 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.

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

3.2. The vp-tree

The structure we use for these experiments is the *vantage point tree* or *vp-tree* (Hjaltason & Samet, 2003). We choose the vp-tree due to its relative simplicity and straightforward application, but our techniques may be easily applied to other MAMs. This is discussed in Section 5.

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, all targets in the target set are sorted according to their 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. Algorithm 1 shows this 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:

$$d_{\mathbf{v}}(\mathbf{t}, \mathbf{q}) = |d(\mathbf{t}, \mathbf{v}) - d(\mathbf{q}, \mathbf{v})| \tag{7}$$

the metric properties of symmetry and the triangular inequality give us

$$d(\mathbf{t}, \mathbf{q}) \geq |d(\mathbf{t}, \mathbf{v}) - d(\mathbf{q}, \mathbf{v})| = d_{\mathbf{v}}(\mathbf{t}, \mathbf{q}) \tag{8}$$

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

$$d_{\mathbf{v}}(\mathbf{t}, \mathbf{q}) \geq \tau \Rightarrow d(\mathbf{t}, \mathbf{q}) \geq \tau \tag{9}$$

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. Hence $d(\mathbf{t}, \mathbf{v}) < m$ for all targets $\mathbf{t}$ in the left subtree. If $d(\mathbf{q}, \mathbf{v}) \geq m + \tau$, then it follows from Equations 7 and 9 that the left subtree of $\mathbf{n}$ may be eliminated from consideration. Similarly, if $d(\mathbf{q}, \mathbf{v}) < m - \tau$ then we may eliminate the right subtree. However, if $m + \tau > d(\mathbf{q}, \mathbf{v}) \geq m - \tau$ then we must do an exhaustive search of all descendants of $\mathbf{n}$.

The choice of the vantage point, then, is crucial to the success of the tree. We would like to choose a vantage point such that few of the distances to its children are close to the median distance. In this work, however, we take an orthogonal approach. Given a tree and the associated vantage points, we will modify the distance function to obtain better performance from the tree. More specifically, we will define a notion of loss and margin associated with the constructed tree, then compute gradients that modify the distance function to decrease the loss and increase the margin.

3.3. Tailoring Distance Functions to vp-trees

We compute the functional gradients against the constructed vp-tree in much the same way as was done in Parker et al. (2006): For each training query, we consider 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

Algorithm 1 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($\mathbf{n}$, $\mathcal{T}$, d)
  $s \leftarrow |\mathcal{T}|$
  if $s = 1$ **then**
    $\mathbf{t}_{\mathbf{n}} \leftarrow \mathcal{T}[0]$
    return
  end if
  $\mathbf{t}_{\mathbf{n}} \leftarrow$ CHOOSE-VANTAGE-POINT($\mathcal{T}$)
  sort $\mathcal{T}$ on $d(\mathbf{t}_{\mathbf{n}}, \mathbf{t}_i \in \mathcal{T})$
  $m_{\mathbf{n}} \leftarrow d(\mathbf{t}_v, \mathcal{T}[s/2])$
  CONSTRUCT-TREE($\mathbf{n}_l$, $\mathcal{T}[0 : s/2]$, d)
  CONSTRUCT-TREE($\mathbf{n}_r$, $\mathcal{T}[s/2 : s]$, d)
end function

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 is not uniform as we travel down the path: Distances 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, we fail to eliminate a subtree containing $|\mathcal{T}|/2$ targets and our cost increases on that order. More specifically, suppose we have 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. The loss function at $\mathbf{t}_j$ is weighted by a factor of $|\mathcal{T}|/2^j$.

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 3.1, and by rebuilding the tree where it is incorrect.

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 $\mathbf{q}_i$ $\{\mathbf{t}_{i1}, \mathbf{t}_{i2}, \dots, \mathbf{t}_{ip}\}$.

We 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 is $m_{ij} - d(\mathbf{t}_{ij}, \mathbf{q}_i)$ if the correct target given $\mathbf{q}_i$ is in the left subtree of $\mathbf{n}_{ij}$ and the same expression in reverse 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 Section 2.2, then:

$$v_{ij} = \begin{cases} 1 & \text{if } \mathbf{t}_i \in \text{ left subtree of } \mathbf{n}_{ij} \\ -1 & \text{otherwise} \end{cases} \tag{10}$$

Stating the margin in a single equation, we have:

$$v_{ij}(m_{ij} - d(\mathbf{t}_{ij}, \mathbf{q}_i)) \tag{11}$$

Again, we borrow the following loss formulation from LogitBoost:

$$\log(1 + \exp(v_{ij}[m_{ij} - d(\mathbf{t}_{ij}, \mathbf{q}_i)])) \tag{12}$$

Recall that d in our sequential alignment setting is simply the sum of the costs of the events in the optimal alignment. Recall also from Section 2.2 that $f(a, b, \mathbf{x}, \mathbf{y}, \mathbf{a})$ is the number of times that character a is replaced by b in the optimal alignment $\mathbf{a}$ of sequences $\mathbf{x}$ and $\mathbf{y}$. Define the following shorthand:

$$f_{ij}(a, b) = f(a, b, \mathbf{t}_{ij}, \mathbf{q}_i, \mathbf{a}_{ij}) \tag{13}$$

With this definition, we can show the distance function as:

$$d(\mathbf{t}_{ij}, \mathbf{q}_i) = \sum_a \sum_b c(a, b) f_{ij}(a, b) \tag{14}$$

And finally, the loss function for a given query $\mathbf{q}_i$ and target $\mathbf{t}_{ij}$ on the path to the correct leaf:

$$\log(1 + \exp(v_{ij}[m_{ij} - \sum_a \sum_b c(a, b) f_{ij}(a, b)])) \tag{15}$$

The cumulative loss L over the entire training set is then expressed as the sum of losses over each training query. The loss for each query is in turn a sum of the losses at each target in the path to the correct leaf. In addition, the loss for a given target in the path is weighted according to its depth in the tree, as discussed earlier in this section.

$$L = \sum_i \sum_j \frac{\log(1 + \exp(v_{ij}[m_{ij} - \sum_a \sum_b c(a, b) f_{ij}(a, b)]))}{2^j}$$

We 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 $\delta_{k+1}(a, b)$:

$$\delta_{k+1}(a,b) = \frac{\partial L}{\partial c_k(a,b)}$$
$$= -\sum_i \sum_j \frac{\exp(v_{ij}[m_{ij} - \sum_{a'} \sum_{b'} c_k(a',b') f_{ij}(a',b')]) v_{ij} f_{ij}(a,b)}{2^j (1 + \exp(v_{ij}[m_{ij} - \sum_{a'} \sum_{b'} c_k(a',b') f_{ij}(a',b')]))}$$

Simplifying the above expression and substituting Equation 14 for compactness yields our final functional gradient expression:

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

To ensure the validity of the constructed vp-tree as changes are made to d, we must check each node to make sure the two child vantage points are still on the appropriate sides of the median value. We do this 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, we send the appropriate halves of the target set to the right and left child nodes and perform the check recursively. If not, we must rebuild the subtree starting at the failed check.

4. Experimental Results

We do experiments in two domains, a synthetic domain and a real-world domain. For each of these domains we compose a set of 2000 possible targets and a corpus of 300 queries, splitting for training and test using 10-fold cross-validation.

For the control approach, we perform 60 iterations of boosting using only the accuracy gradient described in Section 2.2, then construct a vp-tree based on this function. In the experimental approach, we perform 30 iterations of accuracy boosting before constructing a vp-tree. We then perform 30 iterations of boosting using a weighted sum of the accuracy gradient and the efficiency gradient with respect to the constructed tree, as described in Section 3.3. At each iteration of boosting the accuracy of the function is measured, and we assure that the triangular inequality is sufficiently satisfied using the procedure described in Section 3.1. The vantage points for the vp-trees are chosen by randomly sampling the targets at each node and choosing the one for which the set distances to all other targets has the highest variance.

At the conclusion of boosting, the efficiency of both methods is measured using a plot of error tolerance versus the average percent of the target set that is ignored during search. The error here is the number of times we follow the incorrect subtree in the search, unlike in the accuracy plots where error is simply the incorrect retrieval rate. As we decrease τ, our searches progress further and further down the tree, increasing both the ignored targets and the possibility for error. What we would like then, is for the area under this curve to be as large as possible, so that many targets are ignored even at low error rates.

4.1. Synthetic Domain

The first set of experiments is done in the synthetic domain used in Parker et al. (2006) and based on the one in (Joachims, 2003). The constants are modified slightly to make the optimal distance function accurate at large target set sizes. 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 $(0, 29)$. For the target set, we generate random sequences of length 10 from this domain. The queries are generated from random targets, beginning at the first tuple in the selected target, according to the following rules:

1. With probability 0.2, generate a random tuple in the query where $q_2 \geq 25$ (an *insert* event).
2. Else, if $t_2 < 25$, 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).

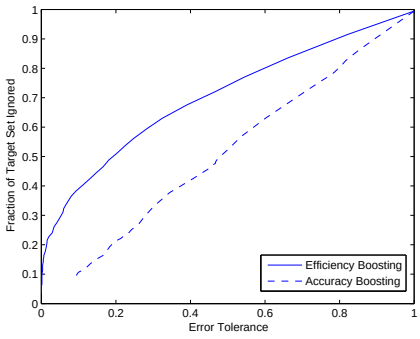

Figure 2. Efficiency in the synthetic domain.

Figure 2 shows the retrieval curves for the synthetic domain. As we can see, the efficiency-boosted distance function has far better retrieval performance than the distance function created with accuracy boosting alone, pruning up to 30% more of the target set during

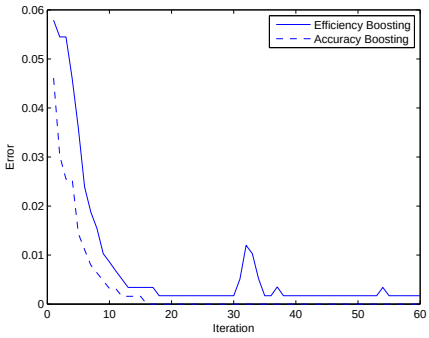

Figure 3. Error rate in the synthetic domain as the distance function is boosted.

search. In Figure 3 we see that, as the distance function evolves over the boosting iterations, the accuracy of the efficiency boosted distance function only falters for a moment when efficiency boosting begins (at iteration 30) but returns to within a fraction of a percent of the error of the accuracy boosted function.

4.2. Query-by-humming Domain

The second set of experiments is done in the so-called "query-by-humming" domain (Meek, 2004; Dannenberg et al., 2003; Pardo et al., 2004). In this domain, the query is an audio file of a person singing or playing a song. The target songs are represented as a database of sequences of notes.

Before use in retrieval, the query must be transcribed to extract a sequence of notes from the raw audio. For a detailed discussion of the transcription process, see Meek (2004). 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:

$$\mathbf{s} = \{(s^p_1, s^d_1), (s^p_2, s^d_2), \ldots, (s^p_{|\mathbf{s}|}, s^d_{|\mathbf{s}|})\}$$

Furthermore, the pitch and duration of the starting event is immaterial as long as the proper *relative* pitches and durations are maintained. Thus, we instead represent the song using *pitch differences* and *duration ratios*:

$$\mathbf{s} = \{(s^\delta_1, s^r_1), (s^\delta_2, s^r_2), \ldots, (s^\delta_{|\mathbf{s}|}, s^r_{|\mathbf{s}|})\}$$

where $s^\delta_i = s^p_{i+1} - s^p_i$ and $s^r_i = \frac{s^d_{i+1}}{s^d_i}$.

These real-valued tuples are binned to give a finite alphabet. Experiments in the literature (Carré et al., 2001; Pardo & Birmingham, 2002) suggest 27 bins for pitch interval and 4 for duration ratio.

The query corpus is a set of roughly 300 queries given by 50 singers on 12 query songs. The training alignments are created using the function given in (Dannenberg et al., 2003). The target set is composed of songs from from the *Digital Tradition* folk song database.

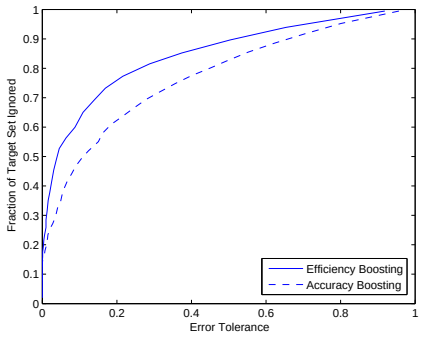

Figure 4. Efficiency in the query-by-humming domain.

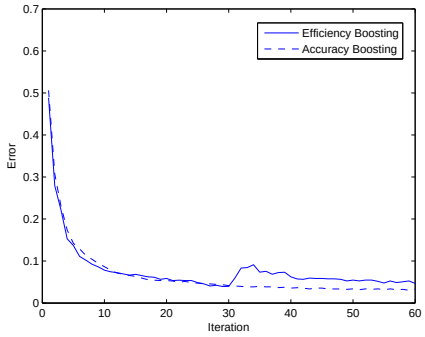

Figure 5. Error rate in the query-by-humming domain as the distance function is boosted.

Figure 4, shows retrieval performance in the query-by-humming domain. Even though the accuracy boosted distance function is already quite efficient, efficiency boosting still gives a 10-20% efficiency increase. Figure 5 again shows an only slight increase in error of about 1-2%.

5. Conclusions and Future Work

We have shown that, using gradient boosting, a distance function in the sequence matching domain can be modified to increase the effectiveness of a metric access method. We have also shown empirically that the gain in efficiency need have little impact on the accuracy of the function. To the best of our knowledge, there have been no attempts in the literature at learning a distance metric specific to an MAM. With these results, we hope that our method can be a point for comparison for future work in this new area.

The obvious direction for this work is expanding it to other MAMs and structure types. To be amenable to these methods, the distance function must be in some sense decomposable, so that the gradient can push parts of it in the proper direction, as is done with the replacement values $c(a, b)$ in these domains.

In order for this work to be applicable to other MAMs, the MAM must be robust to changes in d. A benefit of using vp-trees is that the structure need only be rebuilt where it is invalid after a change to d. In the course of this work, we also derived a gradient expression for cover trees (Beygelzimer et al., 2006), only to find that even small changes to d required rebuilding of the entire tree, thus invalidating previous gradient steps. We also derived a gradient expression for m-trees (Hjaltason & Samet, 2003) which do not require any rebuilding as d changes. Preliminary results with this structure are encouraging.

We note that the efficiency improvements in these domains are somewhat modest. We have early experimental indications that the method will offer even greater improvement on larger datasets, especially those for which the space of possible structures is nearly exhausted in the target set. In these cases, when the efficiency gradient is calculated with complete knowledge of the space of possible structures, we are able to more accurately tune the distance function and achieve better performance.

Finally, it is reasonably easy to extend another discriminative training method, SVM-align (Joachims, 2003), to do what we have done here. Because SVM-align is essentially a constraint optimization method, one can imagine a formalism in which constraints are added for each node in the path of a query, and the learned distance function is an attempt to optimize over these constraints.

Acknowledgments

The authors gratefully acknowledge the support of the Defense Advanced Research Projects Agency under DARPA contract FA8650-06-C-7605 and thank the anonymous reviewers for their comments.

References

Beygelzimer, A., Kakade, S., & Langford, J. (2006). Cover trees for nearest neighbor. *ICML '06: Proceedings of the 23rd international conference on Machine learning* (pp. 97–104). Pittsburgh, Pennsylvania.

Carré, M., Philippe, P., & Apélian, C. (2001). New query-by-humming music retrieval system conception and evaluation based on a query nature study. *Proc. COST G-6 Conference on Digital Audio Effects*. Limerick, Ireland.

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

Faloutsos, C., Barber, R., Flickner, M., Hafner, J., Niblack, W., Petkovic, D., & Equitz, W. (1994). Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, *3*, 231–262.

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

Ghias, A., Logan, J., Chamberlin, D., & Smith, B. C. (1995). Query by humming: Music information retrieval in an audio database. *Proc. 3rd ACM Multimedia Conference* (pp. 231–236).

Hjaltason, G. R., & Samet, H. (2003). Index-driven similarity search in metric spaces (survey article). *ACM Transactions on Database Systems*, *28*, 517–580.

Joachims, T. (2003). *Learning to align sequences, a maximum margin approach* (Technical Report). Cornell University.

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

Meek, C. (2004). *Modelling error in query-by-humming applications*. Doctoral dissertation, The University of Michigan.

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

Pardo, B., Birmingham, W., & Shifrin, J. (2004). 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*.

Parker, C., Fern, A., & Tadepalli, P. (2006). Gradient boosting for sequence alignment. *The Twenty-First National Conference on Artificial Intelligence (AAAI-06)*. Boston, MA.

Skopal, T. (2006). On fast non-metric similarity search by metric access methods. *Proc. 10th International Conference on Extending Database Technology (EDBT '06)* (pp. 718–736).

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

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

Weber, R., Schek, H.-J., & Blott, S. (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proc. 24th Int. Conf. Very Large Data Bases, VLDB* (pp. 194–205).