APPLICATIONS OF BINARY CLASSIFICATION AND ADAPTIVE BOOSTING TO THE QUERY-BY-HUMMING PROBLEM

Charles Parker
Oregon State University
102 Dearborn Hall
Corvallis, OR 97333
parker@cs.orst.edu

ABSTRACT

In the query-by-humming problem, we attempt to retrieve a speci c song from a target set based on a sung query. Recent evaluations of query-by-humming systems show that the state-of-the-art algorithm is a simple dynamic programming-based interval matching technique. Other techniques based on hidden Markov models are far more expensive computationally and do not appear to offer signi cant increases in performance. Here, we borrow techniques from arti cial intelligence to create an algorithm able to outperform the current state-of-the-art with only a negligible increase in running time.

Keywords: melodic retrieval, sequence alignment, arti cial intelligence

1 INTRODUCTION

Most music-related information queries are currently based on meta-data, such as the title or artist of a song. In recent years, we have attempted to create a system in which musical queries could be formed *musically*, giving us, as Downie [1] has said, the ability to query music on its own terms .

One such notion of musical queries is query-by-humming , where a user sings, hums, or whistles a query tune to a computer. The computer has a database of possible songs and attempts to return the song rendered by the user, or perhaps return a ranked list of possible songs. Many possible applications for this have be discussed in a small but growing literature on the problem, ranging from entertainment to commercial to legal. Unfortunately, the problem has proven very dif cult to solve adequately, involving subproblems like pitch detection, note segmentation, polyphonic transcription, and sequence alignment.

It is on the last of these that we will attempt to make progress in this paper. Recent evaluations of various query-by-humming systems have shown that a simple interval-based note matching approach is able to perform as well as other more complex Markov model-based methods when given reasonable queries [2, 3]. Have we reached the best attainable performance on this subproblem?

In this paper, we rst brie y overview query-by-humming and general sequence matching problems. We then review binary classi cation and present a method for constructing alignment models using binary classi cation. Finally, we use *adaptive boosting* to combine our hand-built model with several learned models to create a model that outperforms both on a set of collected test queries. Our concluding remarks indicate improvements for the algorithm and other possible uses for this formalism.

2 BACKGROUND

2.1 Query-by-humming and Sequence Alignment

In the last ten years, the so-called query-by-humming problem has garnered moderate attention in both the information retrieval and the arti cial intelligence literature. Brie y, the problem is as follows: A human sings a query song $\mathbf{q}_i$ that corresponds to some target song $\mathbf{t}_j$ in a *target set* $\mathcal{T}$. If $\mathcal{Q}$ is the space of all possible queries, then our goal is to learn a function f that maps a query $\mathbf{q}_i \in \mathcal{Q}$ to the correct target $C(\mathbf{q}_i) = \mathbf{t}_j$

$$\forall i : f(\mathbf{q}_i, \mathcal{T}) = C(\mathbf{q}_i) = \mathbf{t}_j \tag{1}$$

Often, this is done by creating a function $F : \mathcal{T} \times \mathcal{Q} \mapsto \Re$ so that any (query, target) combination can be given a score, which should be maximized (or minimized) when a query is combined with a matching target:

$$f(\mathbf{q}, \mathcal{T}) = \operatorname*{argmax}_{\mathbf{t} \in \mathcal{T}} F(\mathbf{t}, \mathbf{q}) \tag{2}$$

De ning the function F also gives us a convenient method for *ranking* the targets. The rank of a query in a certain target set is the number of incorrect targets that appear to match the query better than the correct target, given a scoring function. Formally, suppose we have a scoring function F, a query $\mathbf{q}$, and a target set $\mathcal{T}$ containing the correct target $C(\mathbf{q}_i)$. The *rank* of $\mathbf{q}$ under F is:

$$\begin{aligned}&\mathrm{rank}_F(\mathbf{q},\mathcal{T}) = \\ &\qquad \#_{\mathcal{T}}(\mathbf{t}_i | F(\mathbf{t}_i,\mathbf{q}) \geq F(C(\mathbf{q}),\mathbf{q}) \wedge C(\mathbf{q}) \neq \mathbf{t}_i)\end{aligned} \tag{3}$$

This de nition will be used in what follows.

How do we design a good scoring function? Although there are other approaches that merit consideration [4, 5], the prevalent approach in the literature is to convert both target and query into monophonic sequences of notes[1]. These processes are outlined fully in [6] and [2] so we only brie y explain them here. Figure 1 is a summary of this process.

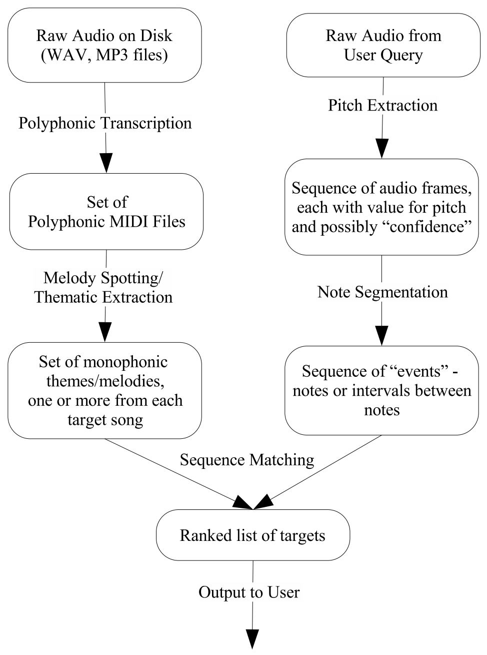
Raw Audio on Disk (WAV, MP3 files)
Raw Audio from User Query
Polyphonic Transcription
Pitch Extraction
Set of Polyphonic MIDI Files
Sequence of audio frames, each with value for pitch and possibly "confidence"
Melody Spotting/ Thematic Extraction
Note Segmentation
Set of monophonic themes/melodies, one or more from each target song
Sequence of "events" - notes or intervals between notes
Sequence Matching
Ranked list of targets
Output to User

Figure 1: A possible general form of a Query-by-Humming system.

For the Query, we begin with raw audio (e.g., in WAV format). We then separate the query into a number of xed-length frames (usually on the order of 0.01 sec.). Each frame is passed to a *pitch detection* algorithm [7] that returns the strongest frequency in that frame along with a *con dence* value indicating the dominance of the strongest frequency. We then sequentially step through these frames, grouping frames of like frequency into notes , using a *note segmentation* algorithm.

For the target, this process is even harder, usually involving the dif cult subproblems of *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). For our experiments, we use a set of monophonic MIDI les in a publicly available database [8] as our target set, and so we avoid this half of the problem.

[1]In [3], the target is instead converted to several monophonic *themes*, each mapping back to the target, but the spirit is generally the same.

Once we have both target songs and query songs represented as sequences, our problem can be restated as a problem in sequence alignment. The standard algorithm for sequence alignment is typically attributed to Smith and Waterman [9] and restated several places in the query-by-humming literature. This algorithm uses *edit costs* to determine the cost of transforming one sequence into another, and uses this to score a target with respect to the query.

Formally, suppose we have a target sequence $\mathbf{t}$ represented as a series of notes $t_1, t_2, \ldots, t_n$ and a query $\mathbf{q}$ represented as a series of notes $q_1, q_2, \ldots, q_m$. Suppose further than we have a *cost model*, K, which gives us $K_i(x)$, the cost of inserting note x into the query sequence, $K_d(x)$, the cost of deleting note x from the target sequence, and $K_m(x,y)$ is the cost of matching the note x in the target to the note y in the query. We can then nd the lowest cost alignment[2] with the following recursion:

$$\begin{aligned}&\mathrm{align}(i,j,\mathbf{t},\mathbf{q}) = \\ &\quad \min \begin{cases} K_m(t_i,q_j) + \mathrm{align}(i+1,j+1,\mathbf{t},\mathbf{q}) \\ K_i(q_j) + \mathrm{align}(i,j+1,\mathbf{t},\mathbf{q}) \\ K_d(t_i) + \mathrm{align}(i+1,j,\mathbf{t},\mathbf{q}) \end{cases}\end{aligned} \tag{4}$$

in which the base case is the end of one or both sequences. With dynamic programming, we can construct a straightforward, ef cient solution. An example[3] of two aligned sequences is shown in Figure 2. We note that K completely speci es the free parameters for this algorithm. For notational convenience, we de ne $\mathrm{align}_K(\mathbf{t},\mathbf{q})$ to be the best alignment of $\mathbf{t}$ and $\mathbf{q}$ with the cost model K. We also de ne $K(\mathbf{a}_{\mathbf{t},\mathbf{q}})$ to be the *cost* of some given alignment $\mathbf{a}$ of $\mathbf{t}$ and $\mathbf{q}$ (as in Figure 2). Finally, we de ne $K(\mathbf{t},\mathbf{q})$ as the lowest cost alignment of $\mathbf{t}$ and $\mathbf{q}$ given the cost model K. To illustrate this notation, consider that, if we use K as the cost model in Equation 4:

$$K(\mathbf{t},\mathbf{q}) = \mathrm{align}(1,1,\mathbf{t},\mathbf{q}) = K(\mathrm{align}_K(\mathbf{t},\mathbf{q})) \tag{5}$$

This may seem like many different ways of saying the same thing, but these notational conveniences will gain more utility in the following sections.

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

Figure 2: A Lexicographical Example of Sequential Alignment.

With a few modi cations to this recursion, we are able to ignore pre xes or suf xes as we see t. There are many variations on this basic theme [10, 11, 12], but all are dependent on nding a cost model that can effectively distinguish the correct target from amongst the incorrect ones.

[2]Notice that we have switched from finding "high scoring" targets to "low cost" targets. This contradicts Equation 2. The reader will please forgive the inconsistency

[3]With thanks to [10].

Although parameters can often be estimated by hand, it would be more useful and practical to estimate these parameters from data. One common method is to construct a table containing each possible note value for K_i and K_d and a table containing the cross product of all possible note values for K_m. If we are then given a series of training alignments (of the form of Figure 2). Then we can simply count the number of times a note is inserted, deleted, or replaced with another note, use these counts to determine probabilities, and use the log probabilities as edit costs [13].

The problem here, as mentioned in [2], is that these tables are so large in this domain that learning them from data becomes intractable. In fact, since both the pitch and the duration of a note are real-valued, it seems we should use a learning algorithm that is able to handle such a representation. A table-based estimate, in other words, may be the wrong hypothesis space for the function we are trying to approximate.

Are there other forms of machine learning that can be used in this context? We turn our attention to this in the next section.

2.2 Binary Classification

To help us learn cost models, we will utilize learning algorithms for binary classi cation. A binary classi cation problem can be stated as a series of duples $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$. In general, $\mathbf{x}_i$ is a vector drawn from some input space X, and $y_i \in \{+1, -1\}$ or $\{\text{positive}, \text{negative}\}$ is the *class label* of $\mathbf{x}_i$. The goal is to learn a function that maps all possible vectors in the input space, even the ones not seen by the learning algorithm, to their correct class label. Formally, if L is the *learning algorithm*, and $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ is the *training data* then $L(T)$ outputs a *hypothesis* $h : X \mapsto \{+1, -1\}$.

Alternatively, we can modify most learning algorithms so that we get a *probability* of the class label rather than the label as output. Our hypotheses will now be of the form $h : X \mapsto [0, 1]$, where a value approaching zero indicates with high probability that the class is negative and a value approaching one indicates the same for the positive class. This is the variant we will use in the sections below.

There are many choices for L. Among them are neural network learners, decision tree learners, and perceptron learners. This is an extremely well-studied problem and can be found in many places in the literature [14]. It is hoped that we can bring some of this knowledge to bear on our current problem.

3 ALIGNMENT EVALUATION VIA BINARY CLASSIFIERS

We now deal with some of the dif culties of learning in this context. Recent attempts at this problem [15] have enjoyed success by formulating the problem *discrimatively* as opposed to *generatively*. In our context, this means to learn the costs with both the correct and incorrect targets in mind. The following subsections outline the method formally.

3.1 The Algorithm

Our discriminative approach is as follows: We take as training data a number of alignments, some of which align a query to a correct target song and some of which align a query to an low-cost, incorrect target song (that is, the best-scoring incorrect target). Each event in both the correct and incorrect alignments (insert, delete, or match) is placed into a set containing all of the events of that type, so that all of the training data is nally in three sets, one for each type of event. Each event is labeled as *positive* if it came from an alignment with a correct target and *negative* if it came from an *incorrect* target.

In each of these sets, we have then a number of positive and negative examples. This is then a binary classi cation problem. The classi er learned from each of these sets will output the probability that a given event came from an alignment of a query with the correct target. This exactly de nes our learned cost model K^L with the three learned classi ers as the functions K_i^L, K_d^L, and K_m^L. A schematic of the algorithm is shown in Figure 3 and psudeocode is given as Algorithm 1.

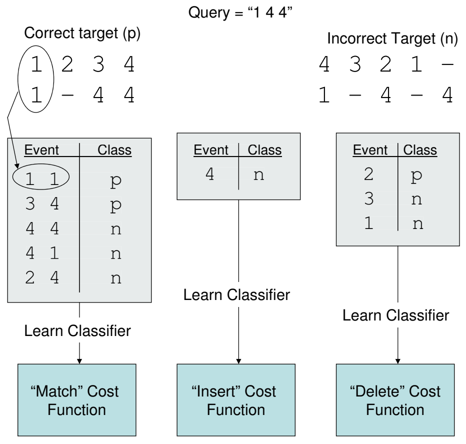

Figure 3: A schematic of our learning process. The 'class' column denotes whether the event came from a correct (p) or incorrect (n) alignment.

The astute reader, observing Algorithm 1, will note that we use a hand-built model to construct the alignment of target and query both in training and after training. Why not just use the learned model to construct the alignment? The reasons here are two-fold:

1. **To reduce running time.** If we have a target of length m notes and a query of length n notes, it takes $O(mn)$ calls to the model to align the two targets and at most $O(m + n)$ calls to the model to evaluate a given alignment. The hand-built model we use here requires only a tiny bit of computation per call, whereas the learned model requires values to be propagated through a learned model (e.g., a neural network), which takes considerably longer. Using the simple model for alignment and the complex one

Algorithm 1 The Binary Classi cation Alignment Algorithm

```
1: Given: A query set $\mathcal{Q}$, a target set $\mathcal{T}$, a cost model $K_0$, and a learning algorithm $L$
2: Initialize training sets $I = D = M = \{\}$
3: for all $\mathbf{q}_i \in \mathcal{Q}$ do
4:     $\mathbf{a}_p \leftarrow \text{align}_{K_0}(C(\mathbf{q}_i), \mathbf{q}_i)$
5:     $\mathbf{t}_n \leftarrow \text{argmax}_{\mathbf{t} \neq C(\mathbf{q}_i)} K_0(\mathbf{t}, \mathbf{q}_i)$
6:     $\mathbf{a}_n \leftarrow \text{align}_{K_0}(\mathbf{t}_n, \mathbf{q}_i)$
7:     for all events $a_i \in \mathbf{a}_p$ do
8:         Label $a_i$ positive
9:         Insert $a_i$ into $I$, $D$, or $M$ if it is an insertion, deletion, or match event, respectively.
10:    end for
11:    for all events $a_j \in \mathbf{a}_n$ do
12:        Label $a_j$ negative
13:        Insert $a_j$ into $I$, $D$, or $M$ if it is an insertion, deletion, or match event, respectively.
14:    end for
15: end for
16: $K_i^L \leftarrow L(I)$
17: $K_d^L \leftarrow L(D)$
18: $K_m^L \leftarrow L(M)$
The nal learned scoring function $F$ is:
    $F(\mathbf{t}, \mathbf{q}) = K^L(\text{align}_{K_0}(\mathbf{t}, \mathbf{q}))$
```

for evaluation considerably reduces running time, especially if we have multiple models as we will see below.

2. **To reduce the difficulty of the learning problem.** Learning in this domain remains dif cult, but training and testing on only high scoring alignments constructed by a reasonable model trims the size of the space, so we only need focus on learning useful concepts that the base model has missed, rather than on aligning sequences in general.

In cases where the learning algorithm is able learn a model that will align correctly and quickly, the hand-built model may be omitted, and random alignments may be used for training. In this application, however, using the hand-built model to align the sequences helps a great deal with performance and speed.

3.2 Particularizing the Algorithm for Query-by-humming

An obvious advantage of this algorithm over table-based methods is that we are not constrained by the number of discrete values that our notes can take. This is handy because to specify a song in terms of its notes, one needs at least two real values per note, a pitch and a duration. This means our characters or symbols in the sequence will be vectors of real values rather than discrete characters as in Figure 3. Learners for binary classi cation problems are by and large comfortable with real-values, unlike the table-based method described above, where real values must in general be rounded or binned .

In what follows below, we assume that notes are vector-valued, containing a component for both the *pitch*, s^p, and the *duration*, s^d, of an event s, so a song $\mathbf{s}$ is represented by a series of duples:

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

Furthermore, the pitch and duration of the starting event is often considered immaterial so long as the proper *relative* pitches and durations are maintained[4]:. Thus, we instead represent the song using *pitch differences* and *duration ratios*:

$$\mathbf{s} = \{(s_1^\delta, s_1^r), (s_2^\delta, s_2^r), \ldots, (s_{|\mathbf{s}|}^\delta, s_{|\mathbf{s}|}^r)\} \quad (6)$$

where

$$s_i^\delta = s_{i+1}^p - s_i^p \text{ and } s_i^r = \frac{s_{i+1}^d}{s_i^d} \quad (7)$$

We make two nal modi cations to the data before passing it to the learning algorithm: We take the log of the duration ratio, as recommended in [17] and for match events, rather than passing in the target and query symbols directly, we pass in the query symbol and the component-wise difference between the two. Table 1 summarizes the features used in training the learned model.

t^δ	-	Pitch difference for target event
$\log(t^r)$	-	Log duration ratio for target event
q^δ	-	Pitch difference for query event
$\log(q^r)$	-	Log duration ratio for query event
Δ^δ	-	$\lvert q^\delta - t^\delta \rvert$
Δ^r	-	$\lvert q^r - t^r \rvert$
Training set for $K_m = \{q^\delta, \log(q^r), \Delta^\delta, \Delta^r\}$		
Training set for $K_i = \{q^\delta, \log(q^r)\}$		
Training set for $K_d = \{t^\delta, \log(t^r)\}$		

Table 1: Table of Features used in Training for the Query-by-humming Problem

4 BOOSTING FOR ACCURATE SELECTION

We have seen then, that if we have a general base model for sequence alignment, then we can use it to learn a new model. However, as we will see in the results section, the learned model performs slightly worse than the base model. A quick scan of the output from these models reveals that they are not making the same mistakes, and that we may reap bene ts if we are somehow able to combine the two.

Furthermore, one could view the algorithm discussed in the previous section as an iterative one. That is, we use the base model to generate the training data for the learning process, learn a new model, then use the learned model to generate new training data for the next iteration. The goal is to train a number of *weak models*, each trained on the mistakes of models in previous iterations. The hope is that these models will combine to give us accuracy greater than any single model alone.

[4]See [16] for an interesting dissenting opinion.

Both of these intuitions, iterative training and weak model combination, are captured in an algorithm known as *adaboost* [18], the adaptive boosting algorithm. Its general form is given in Algorithm 2, and we see the general ideas expressed above. We rst learn a model, then weight the model based on its performance on the training data. We would like the weight, α to increase as the model does better. After learning the model we update a distribution over the training data: Training examples that are misclassi ed get greater weight in the next iteration. These weights are used to force the learning algorithm to focus its attention on the mistakes of previous iterations. We then learn a new model and repeat the process.

Algorithm 2 The general form of the adaptive boosting algorithm.

```
1: Given: A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
2: Initialize training set weights $D_1(j) = \frac{1}{n}$
3: Select learning algorithm, $L$
4: for $i = 1, \ldots, T$ do
5:     Get weak model $h_i \leftarrow L(S)$ using weights $D_i$
6:     Get error $\epsilon_t$ of $h_t$ on $S$
7:     Set $\alpha = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
8:     Update the distribution:
           $$D_{i+1}(j) = \frac{D_i(j) \exp(-\alpha_i y_j h_i(x_j))}{Z_t}$$
       where $Z_t$ normalizes $D$ to be a distribution
9: end for
Output the  nal hypothesis, $H$:
   $$H(x) = \text{sign}\left(\sum_{i=1}^{T} \alpha_i h_i(x)\right)$$
```

The version of adaboost given in Algorithm 2 is used for simple binary classi cation problems with 0-1 loss. In the query-by-humming problem, if we have a database of targets of size n, we have an n-class problem. In addition, the loss function is not 0-1 . Our models are able to produce a ranking of the targets based on the query, and we would like to penalize models based on how low the correct target is ranked. For a cost model, K, we then de ne the reasonable loss function $1 - \frac{1}{\text{rank}_K(\mathbf{q}, \mathcal{T})}$.

Using these guidelines, we modify the adaptive boosting algorithm, given as Algorithm 3. The choice of α in the original algorithm is useful only for binary classi cation problems and will not work for us here[5]. We choose α to be the *mean reciprocal rank* or *MRR* as used in [3]. This is closely related to the loss function de ned above. The update for the distribution over the training set is also tailored to binary problems. We again make a modi cation so that our loss function is re ected in the update. The nal model is simply a weighted combination of the models learned in each iteration.

[5]This is because it uses the fact that a consistently wrong model can be given a negative weight and the wrong predictions made to be right. In our application, there is no such notion of a "mirror image" model

Algorithm 3 The adaptive boosting algorithm applied to learning cost models for alignment

```
1: Given: A query set $\mathcal{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$, a target set $\mathcal{T}$,
   and a base model $K_0$
2: Select learning algorithm, $L$
3: Initialize training set weights $D_1(j) = \frac{1}{n}$
4: for $i = 1, \ldots, T$ do
5:     Learn weak cost model $K_i$ from $\mathcal{T}$ and $\mathcal{Q}$ using
       Algorithm 1, base model $K_0$, and weights $D_i$
6:     Set $\alpha = \frac{1}{n} \sum_{\mathbf{q}_k \in \mathcal{Q}} \frac{1}{\text{rank}_{K_i}(\mathbf{q}_k, \mathcal{T})}$
7:     Update the distribution:
           $$D_{i+1}(j) = \frac{D_i(j) \exp\left(\frac{\alpha_i}{2} - \frac{\alpha_i}{\text{rank}_{K_i}(\mathbf{q}_k, \mathcal{T})}\right)}{Z_t}$$
       where $Z_t$ normalizes $D$ to be a distribution.
8: end for
Output the boosted cost model, $K^*$:
   $$K^*(\mathbf{t}, \mathbf{q}) = \sum_{i=0}^{T} \alpha_i K_i(\text{align}_{K_0}(\mathbf{t}, \mathbf{q}))$$
```

With these changes, we should be able to combine the weak models learned into a single, stronger model. This we will attempt to verify experimentally.

5 EXPERIMENTAL SETUP

As data for our experiment, we use a body of sung queries collected by the author. There are a total of 50 singers and 12 different songs. The singers were we query over a total of 12 songs, with each singer choosing the four they were most familiar with and singing a small, prede ned excerpt from each one. For training we split the data on both singer and song, so the algorithm is tested on singers and songs it has never heard before. This results in a training set of 100 queries over 6 songs and 15 singers and a test set of 321 queries over 6 songs 35 singers. The singers in the experiments were generally amateur singers with experience in a college or church choir but not professionally. We imagine that this demographic will be the most likely users of a nished query-by-humming system.

For the target set, we also split the data, using a 421 song database for training and a 2000 song database for testing. We use the 2000 songs from the test database to simulate databases of smaller target sets by drawing randomly from these 2000 songs. For each simulated size we draw 10 random databases from our base set of 2000 and average the results. The 2421 target songs as well as the 12 query songs are taken from the *Digital Tradition* [8] database of monophonic folk melodies.

Our base, hand-built model is the interval matching model constructed in [3] and shown to be state-of-the-art. The learned and boosted models are constructed using the methods outlined in previous sections. The binary classi er we use is a neural network as implemented in [19] with the default parameters. No tuning was done. The type of alignment used here is the local type de ned in

[10], where we ignore pre xes and suf xes in the target but not the query. That is, the algorithm assumes that the query is contained whole in the correct target.

6 RESULTS

To evaluate our models we plot MRR as the size of the target set size increases in Figure 4 and the percentage of songs ranked rst as target set size increases in Figure 5. We would expect, as the target set grows, that both of these measures will decrease on all models, but that better models will show less of a decrease. For reference, consult [3].

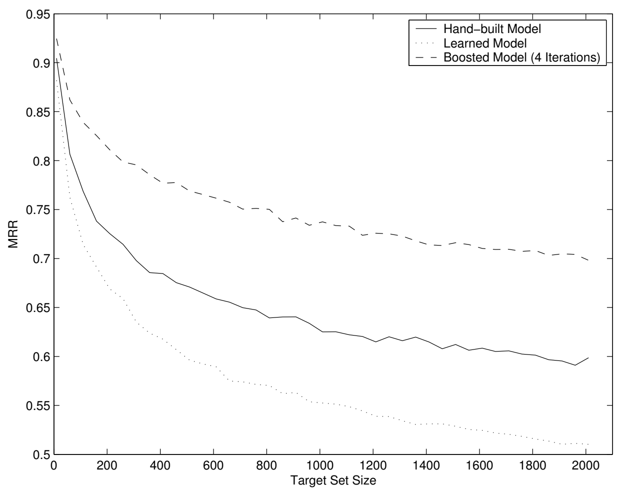

Figure 4: Mean reciprocal rank versus increasing target set size.

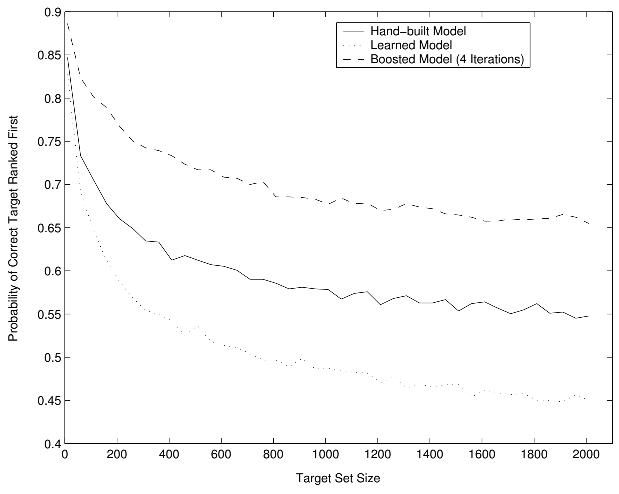

Figure 5: Percentage of queries returning the correct target ranked rst against increasing target set size.

As we can see from the plots, the hand-built interval matching model outperforms the model learned in the rst iteration, but the boosted model is able to outperform both by a signi cant margin, returning the correct target more than 10% more often than the hand-built model.

We also note that there are vast differences in performance between this test of the standard interval model and previous tests. In particular, this test gives a much better MRR than in [3]. We echo the comments made there about results varying wildly with the query set used. We imagine this is due to the fact that the subjects for this test were told exactly what to sing, and were reminded of the tune before singing. Thus, the queries were likely better renderings of the target than was the case in [3].

In Table 2 we show the MRR for a target set size of 2000 for the hand-built model and for each iteration of the boosting algorithm. We see that the boosting algorithm converges very quickly with our training data.

Model	MRR
Base Model	0.598
Learned Model	0.510
Iteration 1	0.698
Iteration 2	0.683
Iteration 3	0.705
Iteration 4	0.698

Table 2: MRR at a target set size of 2000 for all models

Finally, recent comparisons to the hand-built model used here [3, 2, 6] often involve methods that take substantially more computation without showing substantial improvement. We report anecdotally that the difference in running time between our algorithm and the standard interval matching model is less than a factor of two with no optimizations.

7 CONCLUSIONS AND FUTURE WORK

We have here outlined a method for constructing a query-by-humming system that substantially outperforms current state-of-the-art methods with a negligible increase in running time. To accomplish this, we have used a weak model and a standard binary classi cation algorithm, along with a version of adaptive boosting tailored to this particular problem.

Some interesting phenomena were observed as we experimented with various training data. First, as one might expect, if we train the system on poor queries (where the target is sung incorrectly), then it does not perform very well[6]. If we train on a mix of good queries and poor ones, then the boosting algorithm tends to ring , alternately weighting the good and the poor queries heavily, and thus alternately learning good and poor models.

We also concede that we have abused the boosting formalism here in some sense. We have used our hand-built model, h_0 as one of our weak models. The boosting algorithm is provably convergent in the limit of in nite iterations, but if we continue to iterate the algorithm, we will eventually lose the effect of h_0, because it is not generated from the data. Hence we must be careful of over-iterating the algorithm and select a number of iterations that compromises between the usefulness of learning additional models and the usefulness of the base model. In

[6]One might make the comparison to a child with tone deaf parents if one were so inclined.

this instance, the base model turns out to be useful, capturing some things that the learning process does not. One can imagine an instance in which the learning process captures all of the usefulness of the base model. In this case we would only use the base model for constructing alignments and would be free to iterate until convergence.

So we are not tied to the notion of using a hand-built model as part of the nal model. In fact, we are tied down in relatively few ways. The notion of boosting can be applied to any algorithm that is able to learn a model for selecting sequences based on a set of target-query alignments, and there has been much recent work in machine learning on this subject [20, 21, 15]. If we use the binary classi cation formalism, we can choose any representation for notes that suits us, using absolute values of pitch and time, as in [16] or values for pitch and time relative to other elements as in [2]. In addition, we can imagine using *subsequences* of notes as the symbols of the sequence, rather than single notes. This corresponds to the idea of *sliding window classi cation* as outlined in [22]. With parameter tuning, it is highly probable that performance will increase further still. Preliminary experiments indicate as much.

Finally, we note that this formalism is not tied to the query-by-humming problem. This method may be useful for retrieval of structures other than sequences of notes (sequences of other types of elements, trees, graphs, etc.). Future work may entail expanding the ideas outlined in this paper to other domains.

REFERENCES

[1] Stephen Downie and Prof. Michael Nelson. Evaluation of a simple and effective music information retrival method. In *Proc. 23rd International ACM SIGIR conference on Research and Development in Information Retrieval*, 2000.

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

[3] 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.

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

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

[6] 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.

[7] 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.

[8] The Digital Tradition Folk Music Database.

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

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

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

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

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

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

[15] Ioannis Tsochantaridis, Thoman Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proc. 21st International Conference on Machine Learning*, 2004.

[16] 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.

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

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

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

[20] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random elds: Probabilistic models for segmenting and lebeling sequence data. In *ICML*, 2001.

[21] Ben Taskar, Carlos Guestrin, and Daphane Koller. Max margin markov networks. In *NIPS*, 2004.

[22] 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.