Data Min Knowl Disc (2008) 17:402–430
DOI 10.1007/s10618-008-0096-z

Learning to hash: forgiving hash functions and applications

Shumeet Baluja · Michele Covell

Received: 9 January 2008 / Accepted: 23 April 2008 / Published online: 16 May 2008

Abstract The problem of efficiently finding similar items in a large corpus of high-dimensional data points arises in many real-world tasks, such as music, image, and video retrieval. Beyond the scaling difficulties that arise with lookups in large data sets, the complexity in these domains is exacerbated by an imprecise definition of similarity. In this paper, we describe a method to learn a similarity function from only weakly labeled positive examples. Once learned, this similarity function is used as the basis of a hash function to severely constrain the number of points considered for each lookup. Tested on a large real-world audio dataset, only a tiny fraction of the points (∼0.27%) are ever considered for each lookup. To increase efficiency, no comparisons in the original high-dimensional space of points are required. The performance far surpasses, in terms of both efficiency and accuracy, a state-of-the-art Locality-Sensitive-Hashing-based (LSH) technique for the same problem and data set.

Keywords Hashing · Audio matching · Machine learning · Locality sensitive hashing

1 Introduction

This work is motivated by the need to retrieve similar audio, image, and video data from extensive corpora of data. The large number of elements in the corpora, the high

Responsible editor: Eamonn Keogh.

S. Baluja (✉) · M. Covell
Google, Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
e-mail: shumeet@google.com

M. Covell
e-mail: covell@google.com

dimensionality of the points, and the imprecise nature of "similar" make this task challenging in real world systems.

Throughout this paper, we ground our discussion in a real-world task: given an extremely short (∼1.4 s) audio "snippet" sampled from anywhere within a large database of songs, determine from which song that audio snippet came. We will use our system for retrieval from a database of 6,500 songs with 200 snippets extracted from each song, resulting in 1,300,000 snippets (Fig. 1). The short duration of the snippets makes this task particularly challenging. In this presentation, our focus is less on what specific audio features are best for musical similarity (Pampalk 2006; Aucouturier and Pachet 2002) than in providing a method to select from amongst billions of potential features in a way that will be computationally efficient during retrieval.

The task of distortion-robust fingerprinting of music has been widely examined. Many published systems for music fingerprinting attempt to match much longer song snippets, some report results on smaller datasets, and others use prior probabilities of song occurrence to scale their systems (Ke et al. 2005; Haitsma and Kalker 2002; Burges et al. 2003; Shazam 2005). We assume a uniform prior and match extremely small snippets. While our system can easily be incorporated into those designed for longer snippet recognition, by testing on short snippets we highlight the fundamental retrieval issues that are often otherwise masked through the use of extra temporal coherency constraints made possible with longer snippets. Further, we will demonstrate that the improved performance obtained in the retrieval of short snippets also yields substantial improvement in retrieval based on longer snippets.

Finally, a related, but distinct, area of study is that of musical similarity (Pampalk 2006; Aucouturier and Pachet 2002). For these studies in genre classification and recommendation engines, the musical similarity features should provide a similarity metric that generalizes from one song to another. In contrast, for music fingerprinting, the features should distinguish amongst songs (and even amongst renditions of the same song), since the typical goal is assertion of copyright ownership. Audio fingerprinting must handle deliberately introduced distortions, meant to be subtle enough that they are not objectionable to the listener while still rendering the distorted piece

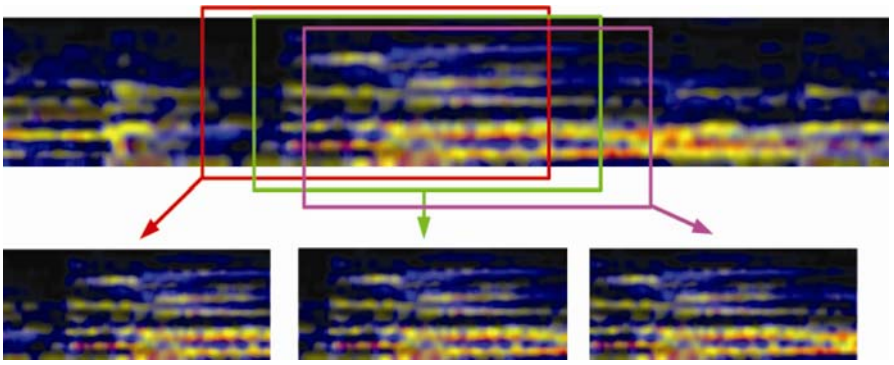

Fig. 1 A typical spectrogram and extracted snippets; note the overlap. The task: given any snippet, find others from the same song

of music difficult to recognize by an automatic classifier. This application area has led research in this area (Burges et al. 2003; Haitsma and Kalker 2002) away from time-domain metrics (e.g., zero crossing rate), where across-critical-band phase variations to could be deliberately introduced to modulate the characteristics. For the same reason (deliberate distortion/degradations to avoid detection), unlike music similarity research, research in audio fingerprinting typically uses a very limited frequency range (300 Hz to 2–3 kHz). One interesting overlap with music-similarity measures is the question of whether the selected features taken from local spectrogram images correspond to some of the metrics for fluctuation patterns described in (Pampalk 2006).

1.1 Background and related work

Generally stated, we need to learn how to retrieve examples from a database that are similar to a probe example in a manner that is both efficient and captures a potentially imprecise notion of similar (Chaudhuri et al. 2003). One way to do this is to learn a distance metric that (ideally) forces the smallest distance between points that are known to be dissimilar to be larger than the largest distance between points that are known to be similar (Hastie and Tibshirani 1996; Shental et al. 2002; Bar-Hillel et al. 2003; Tsang et al. 2005). These methods can be used with k-nearest neighbors ($k$nn) approaches. In fact, $k$nn approaches are well suited to our task: they work with many classes and are able to dynamically add new classes without retraining.

Approximate $k$nn is efficiently implemented through Locality-Sensitive Hashing (LSH) (Gionis et al. 1999). LSH and other hash functions are sublinear in the number of elements examined compared to the size of the database. LSH works for points with feature vectors that can be compared in a Euclidean space. LSH partitions the feature vectors into l subvectors and then hashes each point into l separate hash tables, each hash table using one of the subvectors as input to the hash function. Candidate neighbors can then be efficiently retrieved by partitioning the probe feature vector and collecting the entries in the corresponding hash bins. The final list of potential neighbors can be created by vote counting, with each hash casting votes for the entries of its indexed bin, and retaining the candidates that receive some minimum number of votes, v_t. If $v_t = 1$, this takes the union of the candidate lists. If $v_t = l$,this takes the intersection.

This idea has been extended by (Shakhnarovich et al. 2003) to Parameter-Sensitive Hashing (PSH). To remove the Euclidean requirement, PSH uses paired examples, both positive (similar) and negative (dissimilar) to learn the l hash functions that best cluster the two halves of each positive example while best separating the two halves of each negative example. They create their negative examples by pairing training images that are taken from sufficiently distant regions, in their hidden-parameter space (for their case, joint angles). Their unstated assumptions are two fold: (1) that there is an a priori ordered labeling of the parameter space (e.g., joint angle) that they can use in creating negative training examples to avoid negative examples that are effectively indistinguishable, in both observation and hidden-parameter space, from positive examples and (2) that any hash function that adequately learns their unambiguously negative examples will provide (nearly) equal occupancy on the various hash

bins so that the validation step does not then process unpredictably large numbers of potential neighbors.

The (Shakhnarovich et al. 2003) approach is closest to the one presented here. A major difference is that we do not have a metric parameter space as a given on our training space. A naïve solution to this would be to treat all example pairs that are from distinct songs as equally valid negative examples. This approach would be very similar to the static-label learning approach that we present for baseline comparison. In our work, the learning function is created using only weakly labeled positive examples (similar pairs) coupled with a forced constraint towards maximum entropy (nearly uniform occupancy). We *do not* explicitly create negative examples (dissimilar pairs), but instead rely on our maximum-entropy constraint to provide that separation. As will be shown, our learning constraints have the advantage of allowing for similarities between only nominally distinct samples, without requiring the learning structure to attempt to discover minor (or non-existent) differences in these close examples. Using dynamic relabeling, we implicitly learn an ordering that approaches the continuous parameter-space distance that is already a given in the problem that (Shakhnarovich et al. 2003) solve.

With respect to the application area of music fingerprinting, in Sect. 5, we compare our performance to a previously published, state-of-the-art, approach that is currently in use in large-scale deployment, termed Waveprint (Baluja and Covell 2006). Waveprint uses the same spectrogram parameters as we use in this work and that have been successfully used in previous work (Haitsma and Kalker 2002; Ke et al. 2005): 370-ms long slices taken every 11.6 ms and sampled in 32 bins on the mel-frequency scale, between 300 and 2000 Hz. It then treats the spectrogram as the source of local spectral images by extracting 128 slices at a time (giving 32×128 images). The high degree of overlap, both in the spectrogram calculation and in the spectral image extraction, was designed to provide a relatively high robustness against temporal offsets. Each spectral image is then treated as a query (or as an entry) for approximate image matching. More details of Waveprint, and how it compared to this system, will be given in Sect. 5.

The remainder of this paper is organized as follows. The next section will describe the properties we need for learning a hash function, and describe how we adapted AdaBoost to learn individual bits of our hash functions. In Sect. 3, we illustrate how multiple AdaBoost learners can be used to learn the full hash function. Section 4 shows how the system can be scaled to handle large amounts of data. Section 5 demonstrates the system concretely on a real-world audio-retrieval task. Section 6 closes the paper with conclusions and future work.

2 Learning hash functions

All deterministic hash functions map the same point to the same bin. Our goal is to create a hash function that also groups "similar" points in the same bin, where similar is defined by the task. We call this a *forgiving hash function* in that it forgives differences that are small with respect to the implicit distance function being calculated.

To learn a forgiving hash function for this task, we need to satisfy several objectives: (1) The hash function must be able to work without explicitly defining a distance metric. (2) The hash function must learn with only weakly labeled examples; in our task, we have only weak indications of what points are similar (the song label), but we do not have information of which *parts* of songs sound similar to other parts. (3) The hash function must be able to generalize beyond the examples given; we will not be able to train it on samples from all the songs with which we expect to use it. Effectively, this means that the learned function must maintain entropy in the hashes for even new samples; whether or not the songs have been seen before, they must be well distributed across the hash bins. If entropy is not maintained, points will be hashed to a small number of bins, thereby rendering the hash ineffective.

The requirement to explicitly control entropy throughout training is a primary concern. In the remainder of Sect. 2, we demonstrate the use of a learning procedure, based on AdaBoost (Freund and Schapire 1996), to learn a hash function that can preserve the entropy required. Entropy is explicitly controlled by carefully constructing the target outputs of the training examples. Section 2.1 gives a brief introduction to the version of AdaBoost that is used in this paper, and the features that are computed on the input spectrograms. AdaBoost will be used to learn a single bit; this will be the fundamental component in creating a hashing function that maps a spectrogram into a hash table. Section 2.2 will describe how the target output labels are created. In Sect. 3, we will show how multiple AdaBoost learners are used to learn multiple bits that can be used to create the full hash function.

2.1 Introduction to AdaBoost

The AdaBoost learning procedures are based on the idea of combining multiple *weak learners* to form a single *strong learner*. The only requirement of each weak learner is that it is able to classify points with better than random probability. AdaBoost is an iterative procedure for picking the weak-classifier to add at each step and also setting its associated weight. The final strong classifier is a thresholded linear function of the selected weak classifiers.

We use a version of AdaBoost that has been used in a variety of vision-based domains, such as face detection (Viola and Jones 2001), image retrieval (Tieu and Viola 2000), and image orientation detection (Zhang et al. 2002; Baluja 2007). To do this, we must first create an image of the audio signal to which we can apply these techniques. An important insight provided by (Ke et al. 2005) is that the 1D audio signal can be effectively processed for retrieval as an image when viewed in a 2D time-frequency representation. Ke's state-of-the-art learning system finds features, via an AdaBoost learning procedure, that integrate the energy in selected frequencies over time. Following their findings, we create our spectrograms using parameter settings that have been found to work well in that and previous audio-fingerprinting studies (Haitsma and Kalker 2002). We use slices that are 371 ms long, taken every 11.6 ms, reduced to 32 logarithmically spaced frequency bins between 318 Hz and 2 kHz. One important consequence of the slice length/spacing combination (371 ms slices each 11.6 ms) of parameters is that the spectrogram varies slowly in time, providing

matching robustness to position uncertainty (in time). The use of logarithmical spacing in frequency was selected based on simplicity, since the detailed band-edge locations are unlikely to have a strong effect under such coarse sampling (only 32 samples across frequency). We then extract spectral images, 11.6 × 128 ms long, resulting in snippets that are effectively 1.4 s long. The resulting image is 32 × 128 pixels large. The next image that is extracted is taken 116 ms later (10 slices later).

For analyzing the image, we use the discrete variant of AdaBoost used in (Viola and Jones 2001), which is simple, efficient, and works well in practice. The main steps of the AdaBoost algorithm are shown Fig. 2. Essentially, it is a greedy learner that, at each iteration, selects the best weak classifier for the weighted errors of the previous step. The weight changes in Step 4 are such that the weak classifier picked in Step 3 would have an error of 0.5 on the newly weighted samples; it will not be picked again at the next iteration. Once all the weak classifiers are selected, they are combined to form a strong classifier by a weighted sum, where the weights are related to the reweighting factors that were applied in Step 4.

Given this learning framework, it is necessary to define how the weak classifiers should be constructed. The effectiveness of a weak classifier is measured on its ability to classify each spectrogram into a binary target output (the setting of the target output for each spectrogram will be discussed in the next section). The weak classifiers in this study are simple; the full set of weak classifiers that can be considered at every step are constructed as follows. Given an audio-image, we construct a binary classifier that outputs a +1 if the average intensity of a rectangle (R1) is greater than the average intensity of rectangle (R2) by a threshold amount, T, and 0 if not. A sample classifier is shown in Fig. 3. This type of feature is both simple to use, and can be efficiently

Input: samples (x_1,y_1) .. (x_n,y_n) where x_i are the images and $y_i = 0$ for the type-0 samples and 1 for type-1 samples.

Initialize weights $w_{1,i} = 0.5/F$, $0.5/M$ for $y_i = 0,1$ respectively, where F and M are the number of type-0 and type-1 samples.

For $t = 1,\ldots,T$ (maximum number of weak classifiers to use):

1. Normalize weights $w_{t,i}$ such that $\Sigma_i w_{t,i} = 1.0$
2. For each weak classifier, C_j, see how well it predicts the classification.
   Measure the error with respect to the weights w_i: $error_t = \Sigma_i w_{t,i} | C_j(x_i) - y_i |$
3. Choose the weak classifier (denoted C_t) with the lowest $error_t$.
4. Update the weights:
   if example is classified incorrectly:
   $$w_{t+1,i} = w_{t,i}$$
   else
   $$w_{t+1,i} = w_{t,i}B_t$$
   where
   $$B_t = \frac{error_t}{1 - error_t}$$

The result of the strong classifier is:

$$S(x): \begin{cases} 1: \sum_{t=1}^{T} \log\left(\frac{1}{B_t}\right) * C_t(x) \geq 0.5 \sum_{t=1}^{T} \log\left(\frac{1}{B_t}\right) \\ 0: \quad \text{otherwise} \end{cases}$$

Fig. 2 A boosting algorithm—adapted from (Viola and Jones 2001). The final classifier output is based on a weighted sum of these weak classifiers

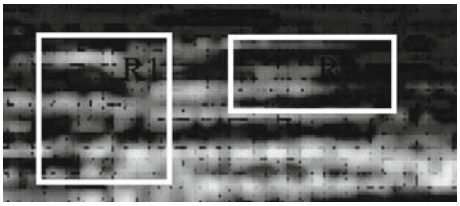

Fig. 3 Typical classifier shown above. For each rectangle, the average intensity is calculated. The classifier returns: (1): if average (R1)-average (R2) > T, (0): if average (R1)-average (R2) ≤ T

calculated by maintaining an integral image, as described in (Viola and Jones 2001). A similar representation is also used in (Ke et al. 2005), in which the basis of feature selection is the discriminative power of the selected region in differentiating between two matching frames (within a distortion set) and two mismatched frames.

One of the time consuming steps in this algorithm is computing the accuracy of all the weak classifiers in each iteration. Given an audio-image of 32 × 128 pixels; the potential number of classifiers (simply counting the combinations of rectangles, R1 & R2) is in the billions, even when symmetries are taken into account. Although the number of candidate classifiers affects only the training-time and not the run-time, evaluating all of the classifiers is not practical. There are a few easy methods to improve the training time. One approach is presented in (Wu et al. 2003)—the error rates for each weak classifier are kept fixed, rather than being reevaluated, in each iteration. Another approach for reducing training times is to randomly select which weak classifiers will be evaluated at each iteration, and select the new classifier from only those that were evaluated (Baluja 2007). At each iteration, the set of classifiers to evaluate is randomly chosen again. Given the expense of even evaluating all the classifiers even once, in the experiments reported here, we used the latter approach.

In each iteration, we evaluated approximately 1,000 of the billions of potential classifiers. For each of the randomly chosen 1000 classifiers, the average output values were calculated for each of audio-images in the training set. Then, 1,000 threshold values, T_i, were calculated to maximize the separation of the target outputs (0 or 1) given the ith-selected classifier's outputs. The resulting classifier is then defined by the triplet {R1, R2, T} (where R1 and R2 are defined by the classifier selection). From the full set of classifiers that were evaluated, the classifier with the lowest error was selected to be part of the set used in the strong classifier.

In the next section, we describe how the values of the target outputs are set; they will be a binary 0 or 1, as needed for the AdaBoost implementation used here.

2.2 Target label assignment & explicit entropy preservation

In order to apply the AdaBoost learning procedure, we must define what the classes of interest are; the learner will be used to discriminate these classes. The output of the AdaBoost classifiers will be used as the basis of the hash function. In this section, we describe how a single bit of the hash function is learned. In the next section, we will describe the procedure for learning multiple bits. Together, the bits will be interpreted as a binary encoded number to index into a position in a hash table.

For training, we will use 1,024 songs. From each of the songs, we select 10 spectrogram images (of approximately 1.4 s that were spaced 116 ms apart). For each *song*, we randomly assign a binary label (either 0 or 1). *Each snippet from the same song is assigned the same label*. In the song label assignment process, we ensure that the number of songs (512) assigned a 0 target is equal to the number of songs (512) assigned a 1 target. The Ada-Boost learner is then trained to reproduce these assignments as the target output when given the corresponding song's spectrogram images as input.

This assignment has two important properties. First, the overall number of 1 and 0 targets is the same; therefore, if the AdaBoost learner has learned the assignment, entropy is preserved in the output. Second, each song's snippets are labeled with the same target. This ensures that when this learned function is used for hashing, similar spectrograms will hash to the same location. For training, therefore, it is important that the snippets chosen from the same song are taken from temporally close regions of the songs—thereby increasing the chances (although not guaranteeing) of being aurally similar.

The results, measured on an independent test set, are shown in Fig. 4. As the number of weak classifiers in the strong classifier increases, we see that very quickly the overall entropy of the target classifications approaches 1.0 (the top line); this reflects the equal number of 1's and 0's that are chosen as target outputs. This is expected and is a good property as entropy is preserved in the outputs.

Since the songs were randomly assigned a 0 or 1, we do not necessarily care about the target output, our concern is only that all the snippets in the same song are classified the same. Therefore, a reasonable criterion is to examine the average entropy of the predicted classifications *within* each of the song's snippets. If the classification worked perfectly, we would expect the entropy to be 0.0; each snippet of the same song would have exactly one label (either a 0 or a 1). However, we see that the entropy remains high. This indicates that the classification has not been learned well. In fact,

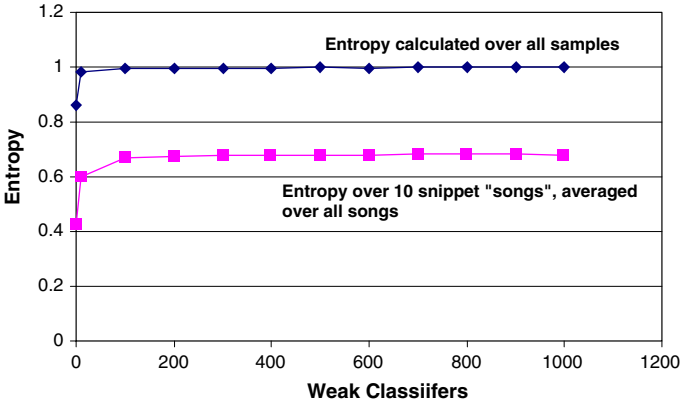

Fig. 4 Note that the entropy of outputs for a strong-boosted classifier across the entire training set remains high, as is desired. This indicates that the distribution of classifications of 0 & 1 are approximately equal; which means that when the outputs are used as hash function, the hash table will be uniformly populated. However, the entropy *within* classes also remains high. We would like the entropy of the labels from all the snippets from a single song to be as close to 0 as possible (indicating that they are all labeled the same)

on average, the entropy magnitude of 0.67 indicates that approximately 8 out of the ten samples were classified as 0 and the other 2 classified as 1 (or equivalently 8 classified as 1, and the other 2 classified as 0). This reflects approximately a 20% error rate. We describe how to reduce this error rate while maintaining high overall entropy.

Recall that in the aforementioned assignment scheme, songs were assigned labels randomly. The drawback of this randomized target assignment is that multiple songs that sound similar may have different target outputs. If we force the learner to make these arbitrary distinctions, we may hinder or entirely prevent it from being able to correctly perform the mapping. Further, although each snippet from the same song is labeled with the target, this is a weak label. Although the snippets that are temporally close may be 'similar', there is no guarantee that snippets that are further apart will be similar—even if they are from the same song. Moreover, snippets from different songs may be more similar than snippets from the same song. If we had a well defined notion of similarity, we could perhaps automatically compute these similarities. Next, we describe an adaptive labeling scheme that overcomes the limitations of the potential discontinuities created by random output labeling.

Instead of statically assigning the target outputs, the targets will shift throughout training. Initially, as before, the target labels are randomly assigned. As training progresses, we measure the incompletely-trained-strong-learner's response to each of the training samples. The response is then averaged across all of the snippets of the same song. The top 50% of the *songs* for which the learner outputs the highest average response (over the song's snippets) are assigned a 1 as the new target. The bottom 50% of the *songs* are assigned a 0 as the new target output. *Note that the top* 50% *of the highest average response songs may not be those that were assigned a* 1, since the training may not be perfect. This is a suitable procedure since it maintains the two properties mentioned earlier: (1) snippets from the same song have the same outputs, and (2) the high-entropy distribution of the outputs, when measured over the entire set, remains. This re-ordering is acceptable since the specific labeling of the examples is not important, only that the above two properties are maintained.

By letting the learner adapt its outputs in this manner, the outputs across training examples are reordered to avoid forcing the learner to make artificial distinctions between potentially similar songs. The outputs are effectively being reordered to perform a weak form of clustering of songs: similar songs are likely to have small distances in the output. Note that if the classifier had been trained to perfection on the training set (i.e. perhaps memorizing/over-training on the training set) this approach may not work, since the target outputs would match the classifier's outputs. More details of a similar dynamic reordering can be found in (Caruana et al. 1996). The training procedure is summarized in Fig. 5. One issue that needs to be addressed carefully is overfitting the training data. There are numerous methods to avoid overfitting; one of the most common is the use of a validation set (Bylander and Tate 2006). Experiments were conducted using a separate validation set; however, because of the simplicity of the classifiers (comparison of the average intensities of the rectangular pixel-regions), the limited number of weak-classifiers used, and the large amount of training data, the strong classifiers did not overfit the data.

The results of the training are given in Fig. 6. In this figure, we see that while the overall entropy of outputs remained high (close to 1.0), the average entropy of the

SELECT M=2^j songs for training	**(Variable Initialization)**
DEFINE $\boldsymbol{B}(t) = t^{th}$ binary code, for $0 \le t < 2^n$	
SET U_t = { $\boldsymbol{B}(t)$ | $0 \le t < 2^n$}	(the binary codes, n is the length of the target output vector)
SET U_s = { S_m | $0 \le m < M$ }	(the unassigned songs)
SET COUNT [t] = 0 for all $t \in U_t$	(track the number of songs assigned to each binary code)
FOREACH (song S_m, $0 \le m < M$)	**(Random Label Assignment)**
FOR all $0 \le x < X_m$, ADD snippets $S_{m,x}$ to training set	
SELECT random t from U_t	
SET TargetLabel$_s$ = t	(TargetLabel is a n dimensional vector).
REMOVE S_m from U_s	
COUNT[t] = COUNT[t] + 1	(Track the number of times each label is used).
If COUNT[t] $\ge$ M / 2^n then REMOVE t from U_t	(Remove label to avoid over assignment).
REPEAT UNTIL TERMINATION	**(Training & Reassignment)**
Reset Strong Learners	
Create n new strong learners with W weak learners each	(The target output for each learner for sample S is the bit n TargetLabel$_s$)
FOREACH (song S_m, $0 \le m < M$)	(Measure the response of the classifiers to each snippet in the song)
SET $\boldsymbol{AvgResponse}_m = \Sigma_{0 \le x < X_m} \mathbf{O}(S_{m,x}) / X_m$	(AvgResponse $_m$ is a n dimensional vector). (where $\mathbf{O}(S_{m,x})$ is the n-dimensional output of the strong-classifiers for snippet $S_{m,x}$)
SET U_t = { $\boldsymbol{B}(t)$ | $0 < t \le 2^n$}	(Begin the reassignment procedure)
SET U_s = { S_m | $0 \le m < M$ }	(the unassigned songs)
SET COUNT [t] = 0 for all $t \in U_t$	
FOR (M iterations)	(For every song do an output reassignment)
FIND $(S_m, t) = \arg\min_{S_m \in U_s, t \in U_t} \| \boldsymbol{t} - \boldsymbol{AvgResponse}_s \|_2$	(Find the binary code most similar to the average outputs).
SET **TargetLabel**$_s$ = t	
REMOVE S_m from U_s	
COUNT[t] = COUNT[t] + 1	
If COUNT[t] $\ge$ M / 2^n REMOVE t from U_t	

Fig. 5 Training algorithm and parameter settings used. Note that the algorithm is described where the number of label outputs can be greater than a single bit. This will be useful in Sect. 3. Settings Used in this study: $j = 10$ (1,024 songs used for training), $n = 1$ (For experiments described in Sect. 2.2, 10 otherwise), W: varied per iteration (as described in the text), $X_m = 10(\forall m)$: 10 snippets were taken from each song for training

predicted outputs within songs decreased dramatically from no reordering (Fig. 4). This is important as it means that snippets from the same song are more likely to have the same outputs. On average, the end entropy magnitude of 0.37 indicates that approximately 9.3 of the 10 outputs were classified as 0 on average, and 0.7 were classified as 1 (or equivalently 9.3 classified as 1, and the other 0.7 classified as 0). This is a significant improvement over the 20% error rate seen with no-reordering.

This is also a significant improvement over the predecessor to this system, that also used dynamic relabeling in the context of neural-network learning (see Fig. 5 of (Baluja and Covell 2007)). With neural-network classification on an identically sized

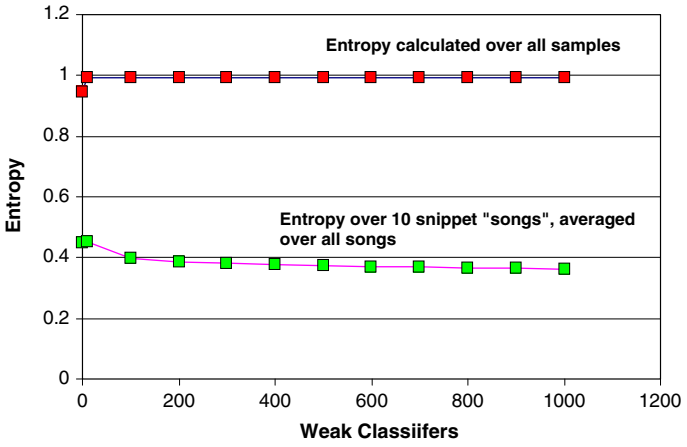

Fig. 6 Note that the entropy of outputs for a strong-boosted classifier across the entire training set remains high, as is desired. This indicates that the distribution of classifications of 0 & 1 are approximately equal; which means that when the outputs are used as hash function, the hash table will be uniformly populated. In contrast to training without reorder (shown in Fig. 4), note that the entropy within songs decreases dramatically

problem (10 outputs to distinguish 1024 songs), error rates without relabeling started at 60% and were brought down to 20% using the same general relabeling approach. With AdaBoost classifiers, our error rates for the same problem start at 20% (without relabeling) and fall to 7% with relabeling. Interestingly, both systems seem to see an improvement factor of approximately 3 times, using dynamic labels over static labels.

3 Towards a full hashing system

To this point, we have described how a single AdaBoost learner can be used to separate a set of songs into two classes. One method of naïvely creating a full hash function using the techniques in the previous section is to repeat the AdaBoost procedure multiple times, from which multiple bits will be output. These bits can then be concatenated and interpreted as a binary code which indexes into a specific bin of a hash table. This simplistic approach, unfortunately, faces a fundamental shortcoming: the multiple learners will likely be correlated when the relabeling procedure is used. If a strong correlation exists between bits of the hash function, only a few of the possible hash bins will be occupied; thereby rendering a hash table ineffective. There are several ways to bias the learners to be independent that can readily be built into the training procedures described thus far:

- Selecting a different set of songs to train with for each learner. This is similar to the ideas used in bagging classifiers (Brieman 1996).
- Randomized output labeling; there is no requirement to label each song with the same label across learners. Randomizing the initial labels may cause different decision boundaries to be created.

- Randomized weak classifier selection: since we only examine a tiny fraction of the possible classifiers in each iteration, randomization in what is learned will be introduced simply by evaluating a different set of classifiers in each run (for example, a different set of rectangles to evaluate on the spectrogram).

Despite employing all three of these mechanisms, the learners that are created using the procedures described to this point are highly correlated. To demonstrate this, we trained 300 strong AdaBoost classifiers. For training each of the learners, 1024 songs were chosen from a total of 5000, the outputs were initially labeled randomly, and the weak-classifier candidate selection throughout training was stochastic. The magnitude of the average pair-wise correlation between 300 independently chosen learners remained very high: approximately 0.6. If used for hashing, this would yield a very sparsely populated hash-table.

To help mitigate the problem of correlation, we can set the outputs of the example of multiple learners with respect to each other. Instead of setting a binary value to the target classification of a song, we now set the labels of a song to length $j = \log_2(M)$ where M is the number of songs that we train from. Each song's label is chosen from a vertex in a j dimensional hypercube. We populate the corners of the hypercube such that an equal number of songs are placed in each corner. We maintain the constraint that all snippets from the same song are given the same length j label. The use of a label of length > 1 is the reason for the generality of the algorithm shown in Fig. 5. In summary, this labeling scheme has three important properties:

- For each bit position in the length j label, 50% of the songs will have a 0 label, the other 50% will have a 1 label, thereby ensuring maximum entropy for bit j.
- For any set of bit positions in the labels, there will be low correlation as we enforce that all of the combinations of possible values are used with equal probability for labeling the training set.
- All snippets of the same song are labeled with the same target; thereby training the close sounding snippets to have the same target outputs, in all of the bit positions.

In every step, the relabeling of the examples maintains these three properties. In relabeling, the adjusted labels are evenly distributed among the songs. Next, we show how the relabeling scheme works on an easily visualized, two dimensional synthetic problem.

3.1 A small example

To visualize the effects of the labeling procedure on training, we show a small example in only two dimensions. In this example, there are 128 songs (shown as clusters of points), composed of 20 points each. These are shown in Fig. 7. We use the label assignment algorithm to train AdaBoost classifiers in the manner described to this point. In Fig. 8a we show the decision boundaries of training 6 independent AdaBoost classifiers with $j = 1$; here, for each AdaBoost learner, 64 songs were assigned to each bit. Figure 8b shows the effects of using $j = 3$ (here 16 songs were assigned to each corner of the hypercube), and Fig. 8c shows the effects of using $j = 7$ (here 1 song was assigned to each corner of the hypercube).

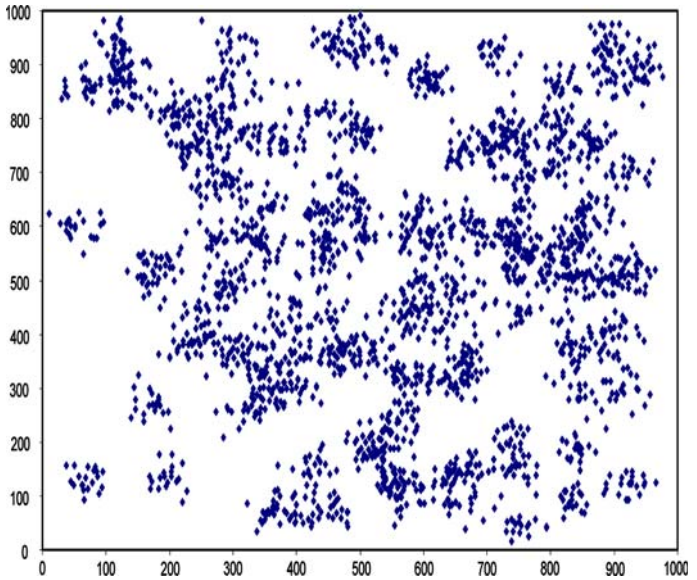

Fig. 7 128 clusters of 20 points each. A small example used to demonstrate the effects of relabeling and simultaneous bit-training

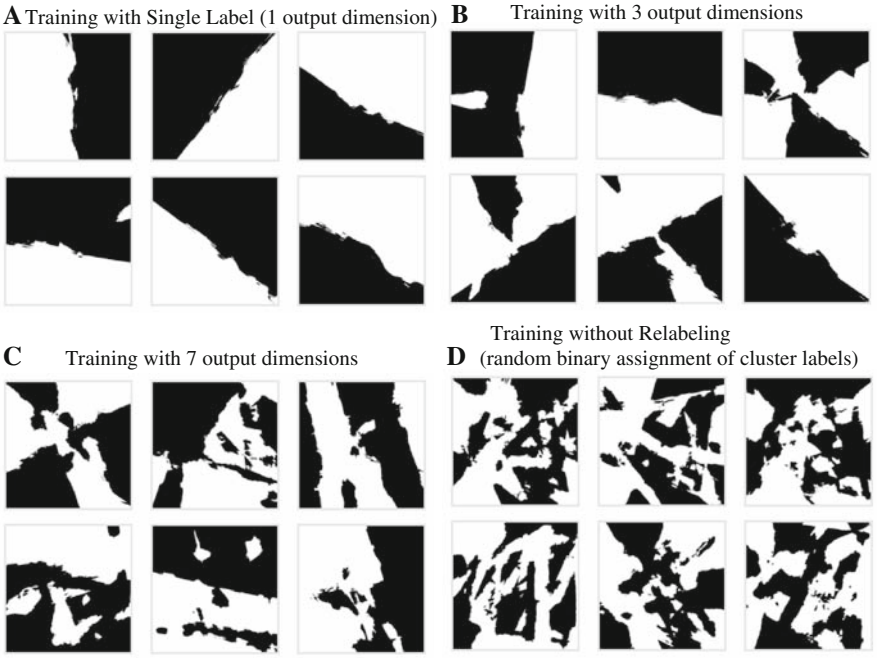

Fig. 8 Trained regions of response for 6 strong learners. (**a**) 6 individual runs when trained with 1-bit each. (**b**) 6 Learners trained through two runs of 3-bits each. (**c**) 6 of the 7 learners trained with 7-bits. (**d**) Trained response of 6 learners when no relabeling is done, for comparison

Table 1 Correlation as label length increases

Use relabel	Length of label (j)	Correlation between 250 classifiers (lower better)	Average final entropy *within each* class (lower better)
Yes	1	0.49	0.003
Yes	3	0.30	0.006
Yes	5	0.20	0.03
Yes	7	0.16	0.10
No	1	0.07	0.30

The images in Fig. 8 show an important aspect of the training. When training with a single label (Fig. 8a), the decision boundaries are of low complexity. However, note that many of the learners, although trained independently, will make decisions that are highly correlated (as measured in Table 1). As the number of output dimensions increases to 3 and 7 (Fig. 8b and c, respectively), the decision boundaries become less correlated, although there is an increase in the complexity shown in this example. Nonetheless, it is suspected that in high dimensions; these types of severe non-linearities may not occur as there will be more dimensions to split the data.

Finally, in Fig. 8d, we examine the results of training without relabeling. Effectively, each cluster is labeled with a random bit. About 6 independent runs are shown. Although these runs will clearly be the least correlated, these classifications will be difficult to learn, as is evidenced in Table 1. Note that this is corroborated by the results seen in the previous section; the within-song entropy remained much higher without relabeling (See Fig. 4) than with relabeling (see Fig. 6). Therefore, although the correlation is low, the effective decision boundaries are *not* reliable for classification.

In summary, we trained approximately 250 learners for each setting of j, and measured the correlation between the learners; these are shown in Table 1. The last column of Table 1 shows the entropy of the classifications inferred by the learners. The smaller the entropy, the more the learner has correctly grouped the snippets together.

3.2 Uncorrelated classifier creation

The previous section provided a simple example of using labels larger than a single bit combined with the relabeling procedures. Here, we employ the techniques described in the previous section to train learners that are uncorrelated for creating song-hashing tools. In a manner similar to the previous section, we train sets of classifiers with $j = 1, 3, 5, 7$ and 10 classifiers respectively. For each setting, we trained a total of 300 classifiers. Each classifier run selected a random 1,024 songs from a possible 5,000. After the classifiers were trained, we measured the correlation and performance of each of the classifiers.

The performance of the classifiers was measured on an individual classifier basis. For each classifier, we measured whether the final-label assigned to snippets (0 or 1)

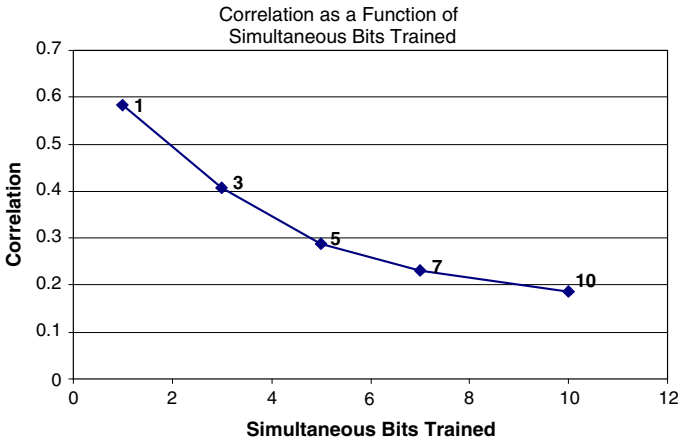

Fig. 9 When training individual bits to be members of the hash function; training each bit individually results in outputs that are correlated nearly 3× greater than when training 10 bits. The correlation reported in this graph is the average correlation of 300 classifiers, measured by correlating $300 \times 299/2$ pairs of outputs

was output by the trained classifier. In each case, the final performance of the classifiers remained uniformly high (above 98% correct), when relabeling was used. The more interesting measurement is the correlation between the learners, shown in Fig. 9. Note that as the number of learners that are simultaneously trained increases, the correlation rapidly drops. This measurement is made across all 300 of the classifiers trained in each setting.

In summary, to this point, we have shown that by dynamically relabeling the training samples, we can find decision boundaries that are easier to learn for the AdaBoost learners than random assignment. Additionally, by ensuring that the label assignment maintains a uniform distribution over possible labels, the overall entropy of the classifier remains high. Importantly, the entropy of the predicted labels *within classes* is lowered when dynamic relabeling is used. Because the actual label of the song is unimportant to the task (only the overall and within-class entropy is important), this type of relabeling is well suited for our task. When extending to multiple classifiers, we found that the correlation between learners is high; which is an undesirable property for using the bits for hashing. We then demonstrated the use of longer length labels (up to length $j = 10$) which are dynamically reassigned to corners of j-dimensional hypercubes during training. Each bit of the label j is learned by a separate AdaBoost learner. Because the target-label assignment process ensures that a uniform number of songs are assigned to each vertex of the j-dimensional hypercube, the high entropy of the labels is maintained, while creating learners that are not correlated with each other.

In the next section, we tie all of our findings together to create the larger hash tables that will be needed in real systems.

3.3 Larger hash tables

In the previous section, we described a mechanism to train multiple bits simultaneously for use as the basis of the hash function. In the examples shown, we trained up to 10 bits simultaneously. Nonetheless, even when each snippet mapping to 10 bits is produced, it only has $2^{10} = 1,024$ unique values, which would yield a tiny hash table. One of the methods to address this is to continue increasing the number of bits trained as was done in the previous section (from 1,3,5,7 to 10 bits). Unfortunately, this scaling would also require a larger number of unique songs as the corners of the j-dimensional-label hypercube that is being trained must be populated with target examples. This increases the training times exponentially as the number of bits increases.

An alternate method of finding more bits is to use the bits from independent training sessions. For example, to create a 20 bit hash function, we could simply concatenate the bits from two 10-bit training sessions. Alternatively, we could select bits from multiple training sessions; we could effectively place all the individual classifiers into a bucket, and select sets of them to concatenate together to create the function to yield the hash address. In this study, we randomly select the bits to use from multiple 10-bit training runs. Specifically, for these experiments, we will conduct 100 10-bit training runs, which will yield 1000 (10×100) total bits to select from. A variety of methods are possible other than random selection to determine which of the 1000 classifiers to use, such as selecting bits that are minimally correlated, or that have minimum mutual information.[1] While we have initially explored these methods, large scale exploration with bit selection methods is left for future research.

Given that we can select bits from multiple training sessions, we are no longer constrained to using 10 bits; we can pick an arbitrary number of outputs from the trained learners as our hash function. Figure 10 shows the performance in terms of the number of candidates in each of the hashed bins and correct matches, as a function of bits that are used for the hash (or equivalently, the number of total hash bins).

As can be seen, by using bits trained from independent 10-bit training procedures, as the number of bits used is increased from 10-bit (1,024 bins) to 22 bits (4,194,304 bins) the average number of elements within each bin decreases dramatically: from 3861 to 35. This reduction in the number of candidates that must be considered is significantly better than was previously seen for dynamically-relabelled neural-networks: our reduction is to 1/72nd of the original candidate-pool size, while the corresponding reduction in (Baluja and Covell 2007) is to 1/18th of the original candidate-pool size.[2] The accuracy of matches (bins containing the same song as the query song) also decreases, from 84% to 59%. The decrease in accuracy is significant, but importantly it is not as large as the efficiency gains. In the next section, we describe how to take advantage of the small number of candidates and regain the loss in matches.

[1] Methods that select bits by explicitly minimizing the mutual information are interesting because these effectively maximize the entropy, which is what is desired since we would like a large number of the hash bins to be occupied.

[2] Note that the numbers in Figs. 6–8 in (Baluja and Covell 2007) are based on a smaller total example population: 10,240 snippets in contrast to 1,300,000 snippets used in this paper. In comparing those results with the results in this paper, the evaluation must be based on ratios of candidates, not total numbers of candidates.

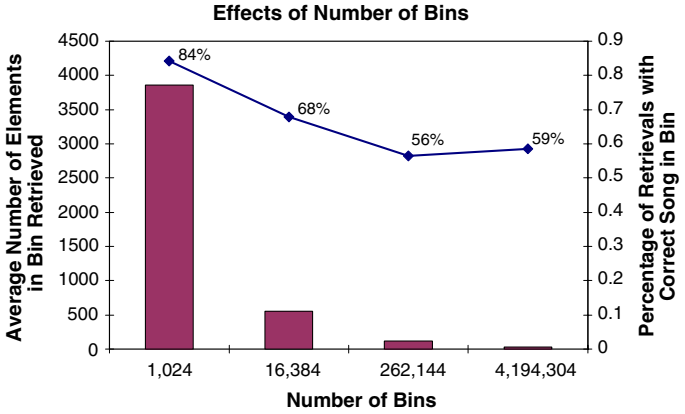

Fig. 10 Using bits trained from independent 10-bit training procedures, as the number of bits used is increased from 10-bit (1,024 bins) to 22 bits (4,194,304 bins) the average number of elements within each bins decreases dramatically: from more than 3600 to less than 50 (bars). The accuracy of matches (bins containing the same song as the query song) also decreases, from 84% to 59% (line). The decrease is significant, but importantly it is not as large as the efficiency gains

4 A complete hashing system

When a new snippet, q, arrives to our system to be looked up, it is passed into an ensemble of trained learners from which selected outputs are used to determine a single hash location in a single hash table. Since we have used a learned hash function that maps similar sounding snippets to the same hash bin, we expect to have resistance to noise. We can further increase our resistance to noise by using multiple hash bins, in a manner similar to locality sensitive hashing (LSH) (Gionis et al. 1999).

We can generalize our approach to l hash tables, with l distinct hash functions. To do this, after building the initial hash-table and function as described to this point, we select from the *unused* bits in the learned ensembles to build another hash function that indexes into another hash table. Now, when q arrives, it is passed through all of the trained learners. The outputs of the learners, if they have been selected into one of $l$hashes, are used to determine the bin of that hash. See Fig. 11. In this section, we will explore the effects of moving from a single hash table ($l = 1$) to many ($l = 24$) hash tables.

In the previous sections, we measured the recall as being whether the hashed bin contained a snippet from the correct song. Here, we tighten the definition of success to be the one we will eventually use in the final system. We rank order the snippets in the database according to how many of the l hash bins (one bin is selected in each of the l hash tables) contained that snippet. The top ranked snippet is the one that occurs most frequently; for example, a perfect match would be a snippet that appeared in all $l$of the hash bins. We count a success if the top-ranked snippet comes from the same song as query q. One of the most important aspects of this procedure is that *we never perform comparisons in the original high-dimensional spectrogram representation.*

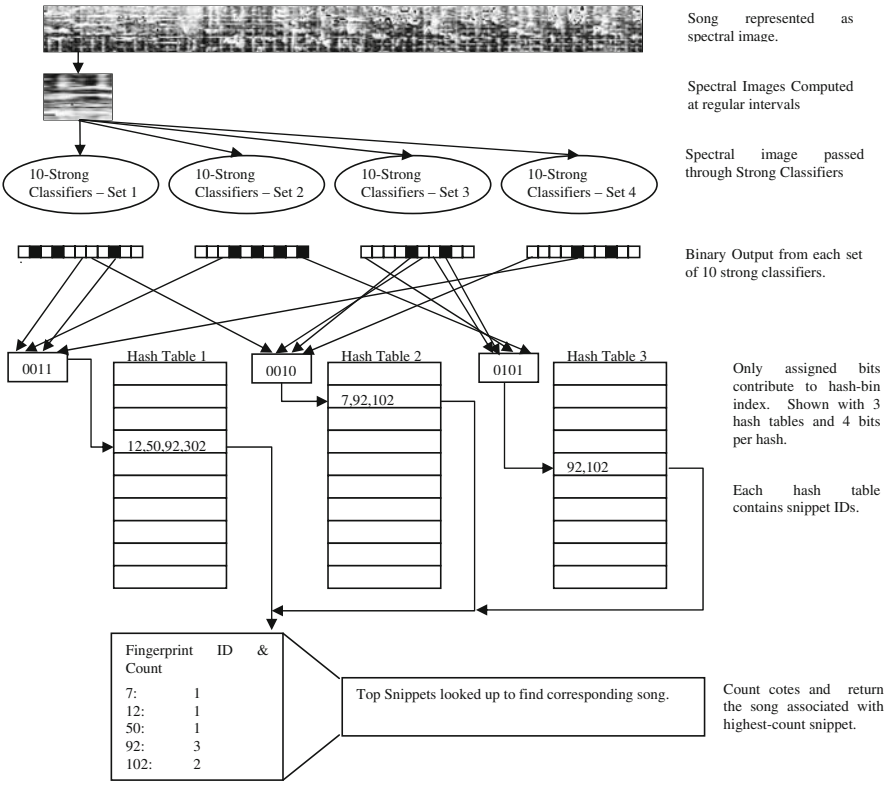

Fig. 11 Overall system architecture with 3 hash tables (in this example, 4 sets of 10-bit learners were employed)

In fact, no snippet representation is kept; it is only represented by recording the hash bins in which it would have been placed.

In the first set of experiments, we examine the effect of increasing the number of hash bins. For this test, we use 6,500 songs with 200 snippets each, a total of 1,300,000 snippets are in the database.[3] For these tests, the individual snippets looked up are removed from the database, and the closest matching snippets retrieved. A trial is considered successful when the most frequently matched snippets (across all hash tables) is from the correct song. Figure 12(left) shows the total number of snippets in the hash bins retrieved for $l = 1$ to $l = 24$ hash tables; this is a measure of efficiency. In the experiments, we also vary the number of bits used in each of the hash tables from 10 (1,024 bins) to 22 (4,194,304 bins). With only 1,024 bins, the number of candidates in the bins is greater than 70,000 (1/20th of the total database) when 24 hash tables are used. With only a single set ($l = 1$) and 1,024 bins, the number of snippets is

[3] For these experiments, 6500 songs were chosen with 200 snippets each. These included the 5000 with 10 snippets each that were used in the training (each trainer was trained on a randomly selected 1,024 subset of the 5000). In later sections, we will run experiments with entirely independent test sets; however, we continue to use this training set here since we are still exploring parameter tradeoffs.

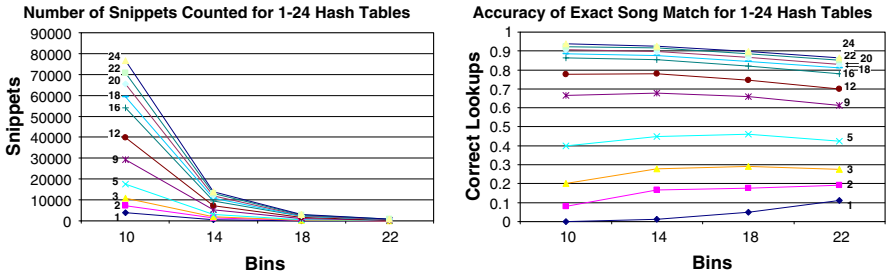

Fig. 12 *Left:* Number of snippets retrieved in the bins of l hash tables (each of the different lines) when varying the number of bins in each hash table (X-Axis); value shown in the number of bits used—the number of bins is 2^x. As expected, as the number of bins increases, the number of elements in each bin decreases. As l increases, the more elements are returned (since the elements in all l bins are returned). *Right:* Accuracy of the most frequently occurring snippet coming from the correct song. There were 1,300,000 songs in the database, representing 200 snippets from 6,500 songs. Chance of randomly guessing correct song is 1/6500

3,861. As expected, looking towards the right of the graph, as the number of bins per hash-table increases, the number of candidates decreases rapidly for all of the settings of l considered, down to about $1/2000^{\text{th}}$ of the total database for 2^{22} bins.

Figure 12(right) provides the most important results; it displays the accuracy of the lookups. For the top lines (24–18 hashes), as the number of bins increases, the ability to find the best match barely decreases—despite the large drop in the number of candidates (as was seen in Fig. 12(left)).[4] Therefore, the system has regained the accuracy lost by using a large number of bins, through the use of multiple hash tables (in a manner similar to LSH).

Again, this approach compares favorably to the previously published results (Baluja and Covell 2007). The percentage of candidates retrieved for any given operating point in the # hashes and # hash bins is consistently lower by a factor of 4 or more. Operating with 22 hash tables (to achieve high recall), we would need to use 2^{18} hash-bins/hash-table under the (Baluja and Covell 2007) approach to have the same candidate pool size as we can achieve with 2^{11} hash-bins/hash-table under the boosting training proposed here. The fewer number of hash bins has an effect on the total memory usage under most implementations. Even ignoring the memory usage implication, using those two operating points as a way to balance the candidate-pool size, our proposed approach does better in top-position ranking, reducing that error from $>10\%$ to $<8\%$.

Finally, in Fig. 12(left), note that as the number of hashes (l) increases, the number of candidates increases almost linearly. This likely indicates that the hashes are largely independent. If they were not independent, the number of unique candidates examined would overlap to a much greater degree. If there was significant repetition of candidates across the hashes, we would see the same "frequent-tie" phenomena that

[4] In Fig. 12-Right, note that for the lowest two lines ($l = 1$–3), as the number of bins increases, the accuracy rises—in almost every other case, it decreases. The increase with $l = 1$–3 occurs because there are a large number of ties (many songs have the same support) and we break ties with random selection. As the number of snippets hashed to the same bin decreases, the correct song competes with fewer incorrect ties and has a higher chance of top rank.

we commented on for $l = 1$: the same songs would be repeatedly grouped and we would be unable to distinguish the correct co-occurrences from the accidental ones.

Some of the interesting operating points in the accuracy/number-of-snippet tradeoffs are listed in Table 2. Note that as the number of bins per hash table increases, the percentage of the database that appears in the hashed-bins decreases. Also note that the accuracy declines rapidly with the smaller number of hash tables. Having more hash tables (in a manner similar to LSH) provides important noise-resistance to our system.

Another measurement that provides insight into the workings of the hashing scheme is to examine how many votes the *top-N* songs received from the lookups in the bins. This gives an indication of how stable the results will be; the larger difference between the top and second-top song, the more stable the results are. As can be seen from Fig. 13 (left and right), there is a sharp decline in the number of votes received for incorrect songs; especially in the case of using 4,194,304 bins, and that is uniformly true across the number of hash-tables ($l = 1$ to $l = 24$) that are used.

It is interesting to contrast this procedure to LSH in terms of how each approach handles hashing approximate matches. LSH hashes only portions of the input vector to each of its multiple hashes. Intuitively, the goal is to ensure that if the points differ on some of the dimensions, by using only a few dimensions per hash, similar points will still be found in many of the hash bins. Our approach attempts to explicitly

Table 2 Accuracy/# of snippets retrieved tradeoffs

Bins per hash table, Hash tables	Accuracy (%)	# Of snippets in retrieved bins	Percentage of DB (%)
2^{10}, 24	94	76,500	5.885
2^{14}, 22	92	13,092	1.007
2^{18}, 24	*90*	*3,003*	*0.231*
2^{22}, 22	85	718	0.055
2^{22}, 16	78	522	0.040
2^{22}, 5	61	317	0.024

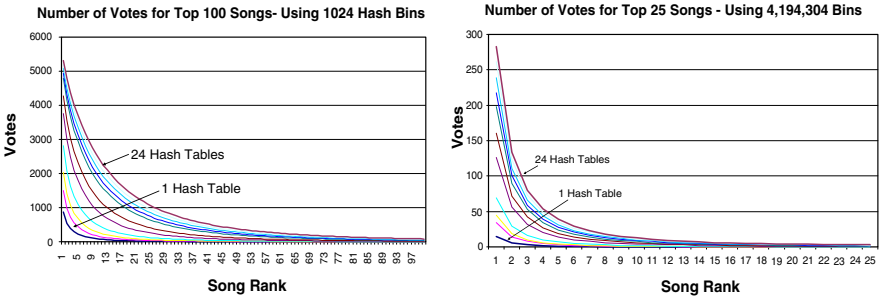

Fig. 13 The number of votes that the *top-N* songs received from the lookups, when varying the number of hash tables (l parameter). A vote is counted for a song-S when a snippet is retrieved from a hash bin that came from song-S. *Left*: When 2^{10} hash bins are used per hash table. *Right*: When 2^{22} hash bins are used. As expected, the number of votes decreases per song when a large number of bins are used, as each bin contains fewer samples. Also note that as l decreases, the number of overall votes decrease as well

learn the similarities and use them to guide the hash function. Further, our approach allows similarity to be calculated on (potentially non-linear) transformations of the input rather than directly on the inputs. The use of multiple hashes in the system used here is to account for the imperfect learning of a difficult similarity functions.

In terms of training times, note that in a simple LSH system, there is effectively no training time since the snippets are directly hashed via random functions. In the proposed system, the training time is dominated by the number of training examples used, the number of bits required in the final system, and the number of weak learners employed for each bit—which is:

$$\begin{aligned}&\mathrm{O}((M \text{ songs}) \times (X \text{ snippets per song}) \times (l \text{ hash-tables})\\&\times(\log_2(\text{hash-bins-per-table})) \times (W \text{ weak learners}))\end{aligned}$$

Note that the exact configuration of the usage of bits, whether more are used per hash-table or more hash tables are used, does not impact the training times. In practice, with non-optimized code, each set of 10 bits used in these experiments required approximately 110 hours on a low-end consumer-level machine with a 2.16 Ghz processor and 2GB memory. For the complete system tested, with 2^{18} bins per hash table and 24 hash tables, 44 sets of 10 bits were trained.

Finally, before continuing to the large-scale experiments, it is instructive to look-back on these results and examine how this complete system would have performed without the re-ordering scheme introduced in this paper. Previously, in Fig. 4, we showed that without re-ordering, the desired entropy properties were not present. Here, we examine the effect of that finding in the context of a complete system. How would a system in which the learners had to learn the mapping from songs to randomly assigned output codes perform? The results are shown in Fig. 14; they corroborate closely the findings from Fig. 4. While the overall number of elements in a bin stay approximately equal for all systems with re-training and without (as expected since the overall entropies in Figs. 4 and 6 are similar), the percentage of trials in which a correct song is found is drastically reduced without re-ordering. This is because the

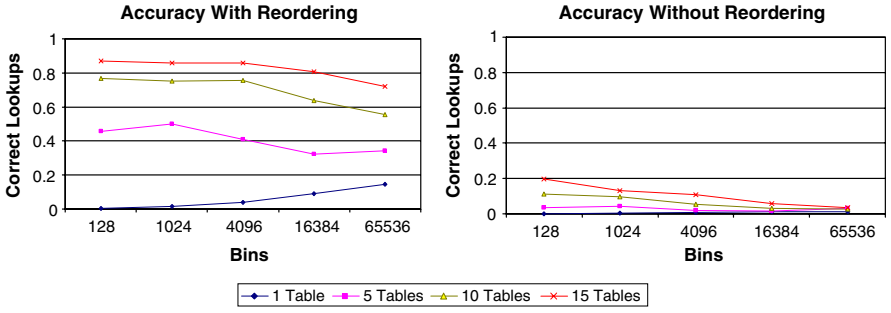

Fig. 14 Accuracy of the most frequently occurring snippet coming from the correct song for systems with re-ordering (*Left*) and without (*Right*) re-ordering. System is measured with (1, 5, 10 and 15) hash tables with (128, 1024, 4096, 16384 and 65536) bins in each table. Note that due to computational constraints, these tests were not conducted on the full-set of samples; nonetheless, both sets systems with and without re-ordering were conducted on the same set, consisting of approximately 50,000 entries, and 5,500 probes

learners are unable to learn the correct mappings well. The random label assignment to songs is an extremely difficult function to learn; despite training, the performance of the learners is quite poor. As can be seen from Fig. 14, some of this poor performance is alleviated through the use of multiple-hash tables. However, the performance stays significantly lower than systems trained with re-ordering; the re-ordering is what makes the function learnable—small distinctions in the input space are re-labeled to small changes in the output space. Without the re-ordering, small distinctions may lead to arbitrarily large differences in the target outputs.

5 Experiments

In the previous section, we concluded our exploration of design parameters, and now use the system in two real-world test cases. We use the hash functions that were learned previously: each AdaBoost classifier was trained with 4,096 weak classifiers. For these tests, 6,500 songs that were never seen before (in any of the previous training or testing) were hashed into 24 hash-tables, each containing 2^{18} bins (this corresponds to the third row of Table 2). Therefore, each signature was 18 bits for each hash table, and there were 24 of them. (In total, the full signature for a snippet was 54 bytes long, although as mentioned earlier, this snippet is never explicitly kept in memory, as a pointer in each hash table is kept in its place). We chose this setting as in the preliminary experiments described in Table 2, it yielded approximately 90% accuracy with the smallest number of lookups.

5.1 Unseen songs

For the first set of experiments, we examine the performance of the system when used with songs that have never previously been encountered. This is an important test; it indicates whether the hash functions that are created by the learners are general enough to be applicable to songs not in the training set. The primary measurement is the simple accuracy of the retrieval: when the set of snippets are retrieved, is the one that is retrieved the most often (from the set of $l = 24$ hash tables) from the correct song. A second measurement is the overall entropy of the resulting hash tables; does it remains high? We measure this by examining the average number of snippets that are in the retrieved hash bins; we hope that we keep the number small.

With 2^{18} bins and 24 hash tables, on an unseen test set, we were able to achieve 89% accuracy (the most frequently retrieved snippet from the 24 hash tables was from the correct song, 89% of the time)—this was measured using simple leave-one-out testing. Importantly, recall that this is for snippets that are only 1.4 s long; this accuracy will be dramatically increased as the snippet length is increased, as will be shown later. The average number of elements found in the bins was 3105.4 ($\sim$0.239% of the 1,300,000 element database). Again we point out that we do not actually compare these elements to the query, we simply count them.

For completeness, to examine the effects of the changing the number of hash-bins per hash tables and the number of hash tables, we experimented with a large parameter set. The results are shown in Fig. 15. There are several trends to note; first, the

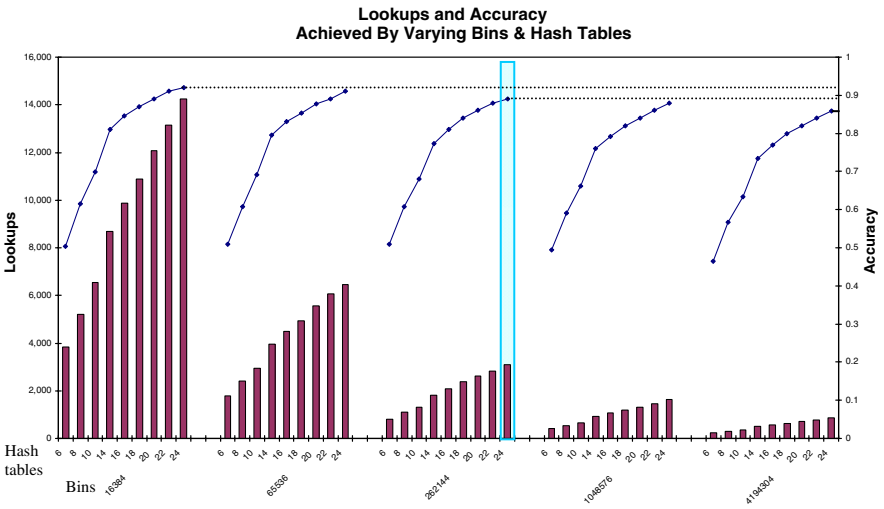

Fig. 15 Bars within each group: as the number of hash tables increases, the number of retrieved samples increases. However, scanning across groups, as the number of bins increases per hash table, the smaller the number of retrieved elements. The lines within each group indicate that as the number of hash tables increases (while keeping the bins constant), there accuracy (right axis) increases. *Most importantly:* as the number of bins increases (scanning across groups from left to right), note that the accuracy for 24 hash tables barely decreases (the thin blue lines)—despite the enormous decrease in the number of points returned (the bars). The highlighted column shows the selected parameter settings

bars, which represent the number of elements found in the looked-up bins, steadily increases within each group; this represents the increase in the number of unique elements retrieved as the number of hash-tables (the l parameter) increases. However, as the number of bins increases, the absolute numbers in the elements retrieved decreases rapidly (in the figure look across the groupings). This decrease in the numbers retrieved is exactly what is expected; the larger the number of bins in a hash table, the more diffuse the elements should be within the table.

Second, the lines, which represent the accuracy of the lookups (how often the most frequently retrieved snippet comes from the correct song), increases as the number of hash tables increases. This is the effect, similar to LSH, where if some set of hashes misplace a point, it is recovered in the remaining hashes. The most important point about Fig. 15 is indicated by the difference in the horizontal dotted lines—although there is a small decline in the top of the blue line for each 24-hash-table point, as the number of bins is increased, the decline in accuracy is small; this provides a good tradeoff for accuracy vs. number of items retrieved: a small tradeoff in accuracy for a very large decrease in the number of items in each bin.

Finally, it is important to put this number in context compared to other large database retrieval schemes. We compare this system with an alternative state-of-the-art LSH-based system called Waveprint (Baluja and Covell 2006). As mentioned in Sect. 1, Waveprint also uses features from the 1.4-s-long 300–2 kHz spectrogram images. However, instead of learning the hash function, as we do here, Waveprint explicitly designs the signature, and thereby the hash functions, using insights from approximate image

retrieval (Jacobs et al. 1995) and from data-stream processing (Gionis et al. 1999). Waveprint uses a *wavelet sketch* to represent query (as well as reference) spectral images. This is done by taking the wavelet transform of the spectral image and then truncating that representation so only the *sign bit* on the top 5% of the largest-amplitude wavelet coefficients remain. This process improves the robustness of the representation to minor changes (e.g., introduction or omission of instrument tracks with the same tempo/evolution as the main-line music) as well as reducing the space requirements for the representation down to 8 K bits without compression. As a final representational transformation, Waveprint uses the insights of data-stream processing (Gionis et al. 1999) to move from this large bit vector representation to a 100-dimensional vector of *min hash* values, with each of these values represented as a single byte (Cohen et al. 2001). Each *min hash* value can be thought of as codeword representing a portion of the original datastream. The min hash values are suited for representing sparse vectors; each value is the occurrence of the first 'on' bit under a random, but static, permutation of the wavelet-sketch. Although the full details of codeword generation in Waveprint is beyond the scope of the paper, it is important to note that the manner in which these codewords are generated guarantee (in a statistical sense) that the Hamming distance between a pair of min-hash signature vectors will approach the Jaccard similarity measure between the original pair of datastream bit vectors. This guarantee means that approximate nearest neighbors can be efficiently and (statistically) reliably found using LSH on the min hash signature. Waveprint uses exactly that property to detect and rank possible matches. Waveprint-based classification (Covell and Baluja 2007) was shown to improve on (Ke et al. 2005), in terms of precision and recall, as well as computational and memory efficiency. Ke's system (available by open-source software (http://www.cs.cmu.edu/~yke/musicretrieval/) and, thus, a reliable point of comparison for Waveprint) was an improvement over the de facto standard of (Haitsma and Kalker 2002).

For our comparisons, we used $\boldsymbol{s}$ sec of a song ($1.4 \leq \mathbf{s} \leq 25$) and integrated the evidence from individual snippet lookups to determine the correct song. Note that these experiments are *harder* than the hardest of the cases expected in practice. Figure 16 illustrates why: even though the (Baluja and Covell 2006) system does not sample the database snippets as closely, their sampling of the probe snippets insures a database-to-nearest-probe misalignment of 23.2 ms, at most, at each database snippet: at least one of the probes will be at least that close, since the probe density is 46.4 ms. By removing the matching snippet from the database, our experiments force *all* database-to-probe misalignments to be a full 116 ms, 5 times more than the closest sampling of (Baluja and Covell 2006). We do not rely on dense probe sampling to guarantee nearby (in-time) matches.

For this test, we want to compare Waveprint and our current forgiving hash system when both systems are set to an approximately equal-computation/-memory operating point. To provide an equal-computation operating point, we used the best-parameter exploration, reported in (Baluja and Covell 2006) and selected an operating point that returned approximately the same number of candidates as were seen by our current system. Results are shown in Table 3.

For the forgiving hasher, we use 2^{18} bins & $l = 24$ hashes. For the Waveprint system, we set the parameters individually for each trial to maximize performance

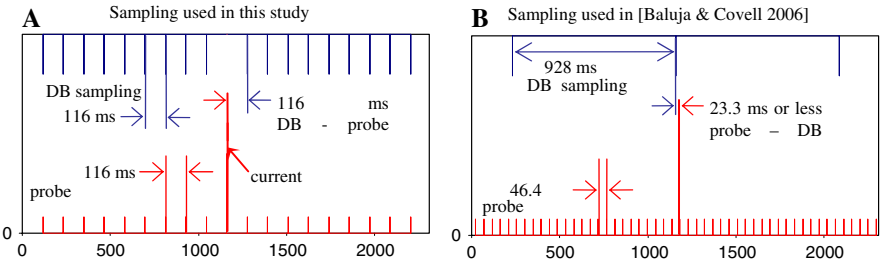

Fig. 16 Sampling used to generate Table 3. The sampling used was chosen to test worst-case interactions between probe and database snippets. For that table, each probe snippet is explicitly deleted from the database before snippet retrieval. The result is the probe snippets are *all* 116 ms away from the nearest database snippet. This test is more difficult than the sampling offsets for (Baluja and Covell 2006), since that system always encounters many snippets with small probe-database separations

Table 3 Percentage of accurately identified songs for query lengths = 1.4, 2,5,13 and 25 s

	Query = 1.4 s (%)	Query = 2 s (%)	Query = 5 s (%)	Query = 13 s (%)	Query = 25 s (%)
Forgiving Hash 2^{18} bins, I=24 hashes	89.5	97.2	99.3	99.3	99.8
Waveprint system	35.5	51.6	73.0	98.4	98.6

Tested with sampling shown in Fig. 16a

while allowing approximately equal hash-table lookups. Unlike (Baluja and Covell 2006), we count the candidates returned from each hash-table, not just the candidates that passed the hash-voting thresholds and thereby triggered retrieval of the original fingerprint. We use this metric on the LSH system since under forgiving hashing, there is no retrieval of an original fingerprint. Each candidate simply has a counter associated with it; neither the spectrogram, nor a representation of the spectrogram, is ever compared with each candidate. In summary, for our system, near perfect accuracy at even 5 s is achieved while considering only a tiny fraction of the candidates.

Looking more closely to the mistakes that are made, it is difficult to make general characterizations of their causes. However, two failure modes repeatedly occurred. The first is silence; many songs contain seconds of no music. Since the detection, at it shortest, is conducted at 1.4 s, these silences are ambiguous. Larger query lengths eliminate this problem. A second failure mode is when songs "sample" other songs—large portions of a song are inserted, sometimes exactly, into another song. As is expected, these cases can confuse the system.

As a final measurement to give insight into system operation, we examine the system performance beyond the single most common snippet found. Beyond looking at the most frequently found snippets for each query, Fig. 17 reveals that there is a smooth drop-off in accuracy of finding the correct song for positions 1–25.[5,6]

[5] This same song can appear in multiple positions in the ranked list since the database contains 200 snippets from each song. Multiple snippets for the same song can be found for each query.

[6] Again, this compares favorably with previous results (Baluja and Covell 2007): the retrieval rates of the top 3 rankings improved by 16–27% in absolute measure (by 22–68% relative measures).

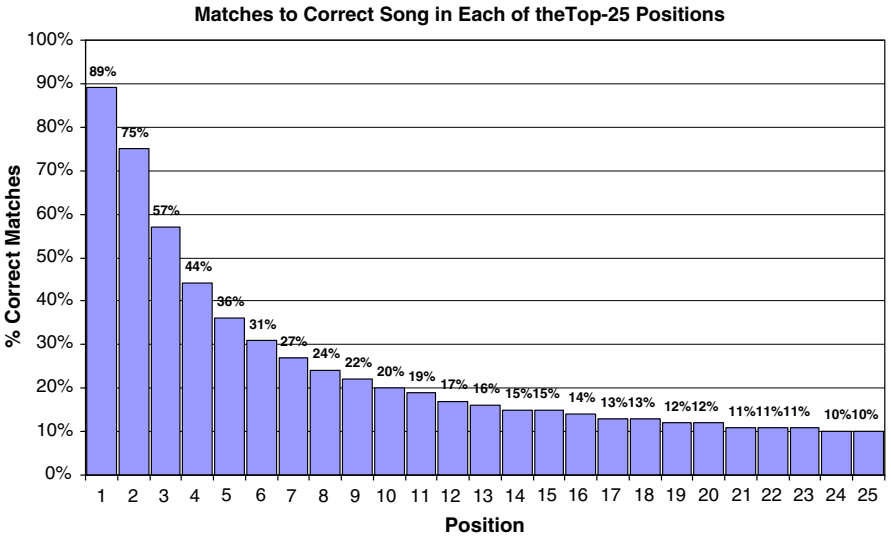

Fig. 17 For each of the top-25 snippets returned, what % of them were from the correct song? 89% of the time, the top snippet returned was from the correct song. The second most frequently returned snippet for each query was from the correct song 75% of the time, etc

5.2 Degraded songs

In addition to the tests described in the previous section, we can also see how the system will perform with degraded copies of the songs as lookups. A standard approach to creating a system that is resilient to degradations is to include, in the training samples, samples that have been degraded in the manners in which degradations are likely to occur in practice. Nonetheless, our hypothesis for the system trained here is that it will be resilient to some types of degradation, even without retraining, since the classifiers were trained to map similar sounding snippets to the same hash bins. To test this, we added an echo to the query song.

The echo was generated as follows: In the original song, we added a single echo. The echo retains 90% of the original signal level and arrives 100 ms after the original sound.[7] The first test conducted replaces each song to be looked-up with one that has echoes. The database remained the same as previous experiments (none of the songs were used in training the classifiers or setting the parameters). In this test, the exact same samples are used for lookups as exist in the database; however, the lookup samples are noisy versions of those in the database. For these tests, 72% of the lookups resulted in samples from the same song. This should be compared with the 89% that was found in the non-degraded songs. Despite a decline in accuracy, it is interesting to note that even with this song degradation, it still significantly outperform the state-of-the-art LSH based system described in the previous section, *with no degradations*.

[7] This effect was created using sox (SOund eXchange) on Linux. The command line parameters controlling the effect were "echo 1.0 0.526 100 0.9" (where 0.526 is the after-echo gain adjustment to avoid clipping).

A second experiment was also conducted with the degraded songs. Instead of using the exact same samples, samples that were taken at a position almost exactly between the database samples were used. These were used to maximally increase the distance to the database samples. Echoes were added to these samples as well. The results did not change; again approximately 72% of the snippet lookups resulted in finding the correct song.

Finally, for both sets of these experiments, a final test was conducted to measure the effectiveness of the retrieval if the complete degraded song was used for lookup. In this case, all of the snippets were allowed a vote for the song that they matched; the song matched the most was considered the final classification. This is similar to the 25 s case found in Table 3. For these experiments, despite being degraded with echoes, and only having been trained on completely clean versions of the songs, 95% of the lookups found the correct song.

6 Conclusions & future work

We have presented a system that surpasses the state of the art, both in terms of efficiency and accuracy, for retrieval in high-dimensional spaces where similarity is not well defined. The forgiving hasher ignores the small differences in snippets and maps them to the same bin by learning a similarity function, based upon the task and the data, that is the basis of the hash function. The learning is based on only weakly labeled positive examples; many of the labels may be inaccurate, and explicitly labeled negative examples are not needed. Not only do our results improve on approaches using static category labels (Ke et al. 2005; Baluja and Covell 2006), it does significantly better than earlier work in dynamic relabeling for better categorization (Baluja and Covell 2007).

In this paper, we tested the system on the task of audio retrieval. The system, in terms of the representation and the algorithms, is designed to work with video and images as well. Currently, these and other domains are under study; the largest required modification to the algorithm is in the set of weak-learners that are employed in the boosting stage of the hash-function creation.

There are many further directions for future work. To ensure high entropy for unseen songs, training with a song set that "spans" the space is desirable. For our experiments, we randomly selected the set of songs; it may be useful to deliberately select a set of songs that cover the space of expected songs more thoroughly. To counter the computational load of using a large set of "spanning" songs for training, it may be possible to train the multiple learners each with a different, small, subset of songs. Another interesting avenue for future exploration is a detailed examination into the song re-ordering that takes place in the output-relabeling phase of the algorithm. It is likely that the songs that are hashed to the same bin are forming an interesting cluster. The type of clustering taking place is based upon the smoothness of the functions created by the learners. This is left for future exploration.

Finally, because the hashing scheme is based upon a learning system, with extra training information, it can be explicitly designed to have smooth degradation properties—i.e. songs that are less similar will map to hash bins that have larger hamming

distances than those that are similar. If this property can be effectively exploited, it will be interesting to look at the set of bins that are clustered together (perhaps with indices that have small hamming distances). This will open the possibility of using forgiving hash approaches in tasks where 'similarity' is even less well-defined. For example, an interesting extension to the task presented here is to use probe-snippets from portions of songs *not* contained in the regions from which the database was constructed. Further, using these techniques for *genre* classification is also a topic of interest, not only in music, but also video and images.

References

Aucouturier J, Pachet F (2002) Music similarity measures: what's the use? In: Proceedings of the 3rd international conference on music information retrieval

Baluja S (2007) Automated image orientation detection: a scalable boosting approach. Pattern Anal Appl 10(3):247–263

Baluja S, Covell M (2006) Content fingerprinting with wavelets. In: Third European conference on visual media production (CVMP), pp 198–207

Baluja S, Covell M (2007) Learning forgiving hash functions: algorithms and large-scale tests. In: International joint conference on artificial intelligence

Bar-Hillel A, Hertz T, Shental N, Weinshall D (2003) Learning distance functions using equivalence relations. In: Proceedings of the twentieth international conference on machine learning

Brieman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Burges JC, Platt JC, Jana S (2003) Distortion discriminant analysis for audio fingerprinting. IEEE Trans Speech Audi Processing 11:165–174

Bylander T, Tate L (2006) Using validation sets to avoid overfitting in AdaBoost. In: Proceedings of the 19th international Florida artificial intelligence research society conference, pp 544–549

Caruana R, Baluja S, Mitchell T (1996) Using the future to "sort out" the present: rankprop and multitask learning. Neural Inf Process Syst 8:959–965

Chaudhuri S, Ganjam K, Ganti V, Motwani R (2003) Robust and efficient fuzzy match for online data cleaning. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, pp 313–324

Cohen E, Datar M, Fujiwara S, Gionis A, Indyk P, Motwani R, Ullman JD, Yang C (2001) Finding interesting associations without support pruning. Knowl Data Eng 13(1):64–78

Covell M, Baluja S (2007) Known-audio detection using waveprint: spectrogram fingerprinting by wavelet hashing. In: Proceedings of the international conference on acoustics, speech, and signal processing

Freund Y, Schapire R (1996) Experiments with a new boosting algorithm. In: Proceedings of the thirteenth international conference on machine learning, pp 148–156

Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: Proceedings of the international conference on very large databases. Edinburgh, Scotland, UK, pp 518–529

Haitsma J, Kalker T (2002), A highly robust audio fingerprinting system. In: Proceedings of International conference on music information retrieval

Hastie T, Tibshirani R (1996) Discriminant adaptive nearest neighbor. IEEE PAMI, 18

Jacobs C, Finkelstein A, Salesin D (1995) Fast multiresolution image querying. In: Proceedings SIGGRAPH

Ke Y, Hoiem D, Sukthankar R (2005) Computer vision for music identification. In: Proceedings of computer vision and pattern recognition, pp 597–604

Pampalk E (2006) Computational models of music similarity and their application to music information retrieval. Doctoral Thesis, Vienna University of Technology, Austria, March 2006

Shakhnarovich G, Viola P, Darrell T (2003) Fast pose estimation with parameter sensitive hashing. In: Proceedings of the international conference on computer vision

Shazam Entertainment (2005). http://shazamentertainment.com

Tieu K, Viola P (2000) Boosting image retrieval. In: Proceedings of computer vision and pattern recognition

Tsang IW, Cheung P-M, Kwok JT (2005) Kernel relevant component analysis for distance metric learning. In: Proceedings of the 2005 IEEE international joint conference on neural networks, vol 2, pp 954–959

Viola P, Jones MJ (2001) Robust real-time object detection. In: Proceedings of the IEEE workshop on statistical and computational theories of vision

Wu J, Rehg J, Mullin M (2003) Learning a rare event detection cascade by direct feature selection. Adv Neural Inf Process Syst 16

Zhang L, Li M, Zhang H (2002) Boosting image orientation detection with indoor vs. outdoor classification. In: IEEE workshop on applications of computer vision