LARGE-SCALE SPEAKER IDENTIFICATION

Ludwig Schmidt[*] MIT

Matthew Sharifi and Ignacio Lopez Moreno Google, Inc.

ABSTRACT

Speaker identification is one of the main tasks in speech processing. In addition to identification accuracy, large-scale applications of speaker identification give rise to another challenge: fast search in the database of speakers. In this paper, we propose a system based on i-vectors, a current approach for speaker identification, and locality sensitive hashing, an algorithm for fast nearest neighbor search in high dimensions. The connection between the two techniques is the cosine distance: on the one hand, we use the cosine distance to compare i-vectors, on the other hand, locality sensitive hashing allows us to quickly approximate the cosine distance in our retrieval procedure. We evaluate our approach on a realistic data set from YouTube with about 1,000 speakers. The results show that our algorithm is approximately one to two orders of magnitude faster than a linear search while maintaining the identification accuracy of an i-vector-based system.

Index Terms— speaker identification, i-vectors, locality sensitive hashing, kd-tree, indexing

1. INTRODUCTION

Speaker identification is one of the core problems in speech processing and acoustic modeling. Applications of speaker identification include authentication in security-critical systems, personalized speech recognition and searching for speakers in large corpora [1]. Due to the increasing amount of data – especially in web-scale applications – fast processing of speech data is becoming increasingly important. While the audio corpus can usually be pre-processed offline and in parallel, the retrieval procedure directly impacts user latency and needs to be executed as quickly as possible. In this paper, we study the problem of fast, text-independent speaker identification in large corpora. Our focus is on maintaining good identification performance while significantly increasing the speed of retrieval. In order to achieve this goal, we combine an i-vector-based speaker identification system with locality sensitive hashing (LSH) [2, 3], a powerful tool for approximate nearest neighbor search in high dimensions.

In this work, we are particularly interested in searching YouTube videos for a given speaker. YouTube is a prime example for the challenges of fast retrieval from a large data set: per day, about 16 years of video are currently being uploaded to YouTube [4]. Even if only a small fraction is human speech, the amount of data to be processed for a single query is still tremendous.

We show that our LSH-based retrieval approach is around $30\times$ faster than a standard linear search on a realistic data set from YouTube with around 1,000 speakers. At the same time, the identification accuracy is still within 95% of the more expensive algorithm. Since we use LSH to approximate the cosine distance of i-vectors, our approach also has provable performance guarantees. Furthermore, there are implementations of LSH-based similarity search for data sets with more than one billion items [5]. Hence our approach promises excellent scalability for large-scale data.

[*]Research conducted as an intern at Google.

2. BACKGROUND

2.1. Speaker identification with i-vectors

Robustly recognizing a speaker in spite of large *inter-session variability*, such as background noise or different communication channels, is one of the main limitations for speaker identification systems. In recent years, this challenge has been addressed with the Factor Analysis (FA) paradigm, which aims to express the main "factors" contributing to the observed variability in a compact way. Initial studies in this direction led to the Join Factor Analysis (JFA) formulation [6], where the acoustic space is divided into different subspaces. These subspaces independently model factors associated with the session variability and factors contributing to the *inter-speaker variability*, i.e., a speaker corresponds to a vector in a low-dimensional subspace.

The JFA model evolved into the Total Variability Model (TVM) [7], where all sources of variability (both speaker and session) are modeled together in a single low-dimensional space. In the TVM approach, the low-dimensional vector of latent factors for a given utterance is called the *i-vector*, and i-vectors are considered sufficient to represent the differences between various utterances. Now, speaker information and undesirable session effects are separated entirely in the i-vector domain. This separation step is typically carried out via classical Linear Discriminant Analysis (LDA) and / or Within Class Covariance Normalization (WCCN) [8]. The cosine distance is typically used for the final comparison of a speaker reference i-vector with an utterance i-vector [7]. Hereafter, we refer to the Total Variability system followed by the classical LDA and WCCN simply as Total Variability or TVM.

More recently, Probabilistic Linear Discriminant Analysis (PLDA) [9] has been proposed to independently model the speaker and session factors in the i-vector space with a probabilistic framework. However, this method performs a more complicated hypothesis test for i-vector matching, which impedes its use with LSH.

2.2. Locality sensitive hashing

The nearest neighbor problem is a core element in many search tasks: given a set of a points $\{x_1, \ldots, x_n\} \subseteq X$, a query point $q \in X$ and a distance function $d : X \times X \to \mathbb{R}^+$, find the point x_i minimizing $d(x_i, q)$. While efficient data structures for the exact problem in low-dimensional spaces are known, they have an exponential dependence on the dimension of X ("curse of dimensionality"). In order to circumvent this issue, LSH offers a trade-off between accuracy and running time. Instead of finding the exact nearest neighbor, the algorithm can return an approximate nearest neighbor, with the retrieval

time depending on the quality of the approximation. An approximation guarantee is still useful because the distance function d is often only an approximation of the ground truth. A particular strength of LSH is its provably sublinear running time, which also holds in practice and has led to many applications of the algorithm [3].

In order to use LSH with a given distance function d, the algorithm relies on a family of *locality sensitive hash functions*. Intuitively, a hash function is locality sensitive if two elements that are close under d are more likely to collide. There is a large body of research on locality sensitive hash functions for a wide range of distance metrics, including the Euclidean distance [10], Jaccard index [11] and the cosine distance [12].

Given a family of locality sensitive hash functions, the LSH algorithm builds a set of hash tables and hashes all points x_i into each hash table. For each hash table, we concatenate several locality sensitive hash functions to avoid unnecessary collisions (boosting precision). We maintain several hash tables to increase the probability of finding a close neighbor (boosting recall). Given a query point q, we look through all hash tables to find the x_i colliding with q and then return the best match.

2.3. Related work

There is a large body of work on using LSH for audio data. A common use case is efficiently finding remixed songs or near-duplicates in a large corpus [13, 14, 15, 16, 17, 18]. These systems are based on low-level acoustic features such as MFCCs and then use a variety of LSH-related techniques, e.g. shingles and min-hashing. While this approach works well for finding near-duplicate audio data, we are interested in text-*independent* speaker identification and hence require a more sophisticated acoustic model.

A recent paper [19] uses LSH for speaker identification with privacy guarantees. The authors mention efficiency gains due to LSH but use cryptographic hash functions and focus on the security aspects of their work. The speaker identification step is based on Gaussian mixture model (GMM) super-vectors without factor analysis while our work uses i-vectors. Moreover, the authors study the performance of their system on the YOHO data set [20], which consists of 138 speakers and is primarily intended for text-*dependent* speaker authentication.

The most closely related work is [21]. While the authors also employ factor analysis in their acoustic modeling step, their utterance comparison model [22] is different from i-vectors. Importantly, the authors use kernelized-LSH [23] in order to implement the distance function implied by their model. In contrast, we use the cosine distance, which is known to give good performance for i-vectors [7] and also provides provable guarantees in conjunction with LSH [12]. Furthermore, the authors of [21] explicitly state that the objective of their paper is to investigate the performance of their method on close-talk microphone recordings with *matched conditions*, leaving a more robust variant that is resistant to noise for further study. In our evaluation, we use a set of videos from YouTube that was not recorded for speaker identification. In particular, the channel effects in some videos are significantly different and noise is present in most recordings.

3. LOCALITY SENSITIVE HASHING FOR SPEAKER IDENTIFICATION

For clarity, we describe our proposed system as two separate components: the generation of i-vectors and the fast retrieval of similar i-vectors using LSH.

3.1. Generation of i-vectors

Given an utterance for which we want to generate an i-vector, we first represent the utterance in terms of a large GMM, the so-called Universal Background Model (UBM), which we parametrize with λ. Formally, let $\Theta = (o_1, \ldots, o_a)$ with $o_i \in \mathbb{R}^D$ be a sequence of spectral observations extracted from the utterance. Then we compute the accumulated and centered first-order Baum-Welch statistics

$$N_m = \sum_t P(m|o_t, \lambda)$$

$$F_m = \sum_t P(m|o_t, \lambda)(o_t - \mu_m) \ ,$$

where μ_m is the mean vector of mixture component m, $m = 1, ..., C$ ranges over the mixture components of the UBM and $P(m|o, \lambda)$ is the Gaussian occupation probability for mixture m and observation o. Hereafter, we refer to $F \in \mathbb{R}^{CD}$ as the vector containing the stacked statistics $F = (F_1^T, \ldots, F_C^T)^T$.

We now denote the i-vector associated with the sequence Θ as $x \in \mathbb{R}^d$. According to the TVM model, the vector F is related to x via the rectangular low-rank matrix $T \in \mathbb{R}^{CD \times d}$, known as the TVM subspace:

$$N^{-1}F = Tx \ ,$$

where $N \in \mathbb{R}^{CD \times CD}$ is a diagonal matrix with C blocks of size $D \times D$ along the diagonal. Block $m = 1, .., C$ is the matrix $N_m I_{(D \times D)}$.

The constraints imposed on the distributions of $P(x)$ and $P(F|x)$ lead to a closed-form solution for $P(x|F)$. The i-vector is the maximum a posteriori (MAP) point estimate of this distribution and is given by

$$x = (I + T^T \Sigma^{-1} N T)^{-1} T^T \Sigma^{-1} F \ ,$$

where $\Sigma \in \mathbb{R}^{CD \times CD}$ is the covariance matrix of F.

Therefore, our i-vector extraction procedure depends on the utterance data and the TVM model parameters λ, T and Σ. We refer to [24] for a more detailed explanation of how to obtain these parameters using the EM algorithm.

If the true speaker labels for each training i-vector are known, the final speaker i-vector is normally obtained by averaging all i-vectors belonging to the same speaker. Since we are interested in an unsupervised setting such as YouTube where speaker labels are not available for most of the utterances, we do not perform this i-vector averaging step in our system and instead keep the i-vectors of all utterances.

3.2. Locality sensitive hashing with i-vectors

As noted in the introduction, the main goal of our work is enabling *fast* retrieval of speakers. In the context of i-vector-based speaker identification, this means the following: for a given query i-vector, we efficiently want to find the best match in our previously computed set of i-vectors. Since this task is an instance of the nearest neighbor problem introduced above, we use LSH in order to enable fast retrieval.

A crucial point when using LSH is the right choice of distance function d. For i-vectors, it has been shown that the cosine distance $d(x, y) = \frac{x \cdot y}{\|x\| \, \|y\|}$ gives very competitive performance for speaker identification [7]. Since the cosine distance can also be approximated well with locality sensitive hash functions [12], we use the cosine distance in our LSH algorithm. In particular, we use hash functions of the form

$$h_r(x) = \begin{cases} 1 & \text{if } x \cdot r \geq 0 \\ 0 & \text{if } x \cdot r < 0 \end{cases} ,$$

where we choose r as a random Gaussian vector. Geometrically, this hash function can be seen as hashing with a random hyperplane: r is perpendicular to the hyperplane and the result of the hash function indicates on which side of the hyperplane x lies. Since r has an isotropic distribution, we have $P[h_r(x) = h_r(y)] = 1 - \theta(x, y)/\pi$, where $\theta(x, y)$ is the angle between vectors x and y.

Our data structure has two main parameters: l, the number of hash tables, and k, the number of hyperplanes per hash table. Let $H_1, \ldots, H_l$ be the hash tables in our data structure. We use the construction from [25] in order to reduce the number of hash function evaluations: we maintain $m \approx \sqrt{l}$ hash functions of length $\frac{k}{2}$ and use the $\binom{m}{2} \approx l$ combinations as hash functions for the l hash tables. Formally, let $u_i(x) = (h_1^i(x), h_2^i(x), \ldots, h_{k/2}^i)$ for $i \in \{1, \ldots, m\}$ and h_j^i sampled as described above. Then the final hash functions for the l hash tables are $h_i(x) = (u_a(x), u_b(x))$ for all choices of a, b such that $1 \leq a < b \leq m$. Hence each h_i hashes an i-vector x to a string of k bits. Note that we do not need to store a full array with 2^k entries for each hash table but can instead resort to standard hashing for large k.

For a given database of i-vectors $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$, we initialize our LSH data structure as follows: each i-vector x_i is hashed with each hash function h_j and then inserted at position $h_j(x_i)$ in hash table H_j. The overall time complexity of the initialization step is $O(ndk\sqrt{l} + nl)$.

Algorithm 1 describes the fast retrieval procedure. The evaluation of the m hash functions u_i in lines 2 and 3 can be efficiently implemented with a vector-matrix multiplication as follows: we stack the normal vectors of the hyperplanes as rows into a matrix $U \in \mathbb{R}^{mk/2 \times d}$. The bits used in the hash functions are then given by $\frac{\operatorname{sgn}(Ux)+1}{2}$. The running time of the retrieval procedure is $O(dk\sqrt{l} + l + M)$, where M is the total number of matches found. M is typically small if the parameters k and l are properly chosen.

Algorithm 1 I-vector retrieval with LSH

```
1: function RetrieveIvector(q)
2:     for i ← 1, ..., m do
3:         Evaluate u_i(q)
4:     C ← {}                                  ▷ Set of candidates
5:     for i ← 1, ..., l do
6:         C ← C ∪ H_i[h_i(q)]                 ▷ Add candidates
7:     return arg min_{x∈C} (x·q)/(‖x‖ ‖q‖)    ▷ Return best candidate
```

4. EXPERIMENTS

For our experiments, we focus on the main application outlined in the introduction: searching for speakers on YouTube.

4.1. Data set

We built a data set from the Google Tech Talk channel on YouTube [26], which contains about 1,803 videos of talks given at Google. Many of the videos have speaker labels and contain one main speaker. We decided on this data set because we wanted to use YouTube data with good ground truth information while avoiding manual labeling and speaker diarization issues.

After removing short videos and videos with more than one speaker, our data set contains 1,111 videos with 998 distinct speakers, with each video containing at least 30 minutes of the corresponding talk. 74 speakers appear in at least two videos. The recording quality of the videos varies with respect to audience noise, equipment, room acoustics and the distance from the speaker to the microphone. The list of videos with a unique ID for each speaker is available online at `http://people.csail.mit.edu/ludwigs/data/youtube_speakers_2013.txt`.

4.2. Results

We conducted two types of experiments. For each type of experiment, we studied the performance of our algorithm for utterances of length 10, 20 and 60 seconds. The utterances were selected randomly from the videos and correspond directly to segments of the talk. Hence, the duration of speech per utterance can be less than its nominal duration.

10tests In this setup, we divide each video into utterances of length t and then select 10 random utterances from each video to form the query set. The remaining utterances are used for i-vector training and as the retrieval database.

holdout10 For each speaker with at least two videos, we select one video randomly as the source of query i-vectors. All remaining videos are used for i-vector training and as the retrieval database. We then extract 10 random utterances of length t from the query videos and use the corresponding i-vectors as query points.

The second experiment setup is significantly more challenging as the speaker identification system has to ignore the channel mismatch between different videos. We include results for the first type of experiments to study the performance of our LSH data structure under matched conditions.

Since the goal of our work is fast retrieval, we focus on the trade-off between identification accuracy and computational efficiency. We use a linear search as comparison baseline and measure the following quantities:

Identification accuracy We count a query as identified correctly if the speaker corresponding to the query i-vector is the same as the speaker corresponding to the i-vector returned by the retrieval algorithm. We only consider the single top-scored speaker returned by the retrieval algorithm. The identification accuracy is then the empirical probablity of correct identification.

Retrieval time We measure the time the retrieval algorithm takes to return a candidate i-vector for a given query i-vector. Note that we exclude the time for converting a query utterance to a query i-vector. Since the time complexity of the conversion step is independent of the size of the database, the i-vector retrieval step will dominate the overall computational cost for large data sets.

Table 1 shows the key details of our experiments. In all experiments, we used 200-dimensional i-vectors. Figures 1 and 2 illustrate the trade-off between accuracy and performance we can achieve by varying the parameters of our LSH data structure. All experiments were conducted on a 3.2GHz CPU and each data point was averaged over all query points, with 10 trials per query point.

The results show that we can achieve speedups by one to two orders of magnitude while sacrificing only a small fraction of the identification accuracy. For an utterance length of 20s, our retrieval algorithm is roughly 150 times faster than the baseline on the 10tests experiments and about 35 times faster on the holdout experiments. In both cases, the relative accuracy is about 95% (we define relative accuracy as the ratio (LSH accuracy)/(baseline accuracy)). Moreover, we can achieve a wide range of trade-offs by varying the parameters of our LSH data structure.

An interesting phenomenon in our results is the performance gap between the matched and unmatched settings (10tests and holdout10, respectively). This discrepancy is probably due to the fact that the i-vectors are better clustered under well-matched recording conditions and consequently, the approximate guarantees of LSH have less impact. Therefore, considering only a small number of candidates in the hash tables is sufficient to find a correct match. In

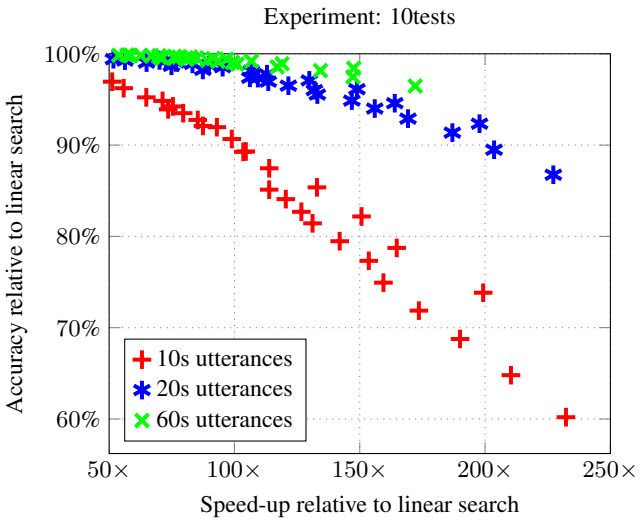

Fig. 1. Speedup vs. accuracy trade-off for the 10tests experiments. Each point corresponds to a choice of parameters for our LSH data structure. We vary k (the number of hyperplanes) from 6 to 20 in steps of 2 and l (the number of hash tables) from 80 to 300 in steps of 20.

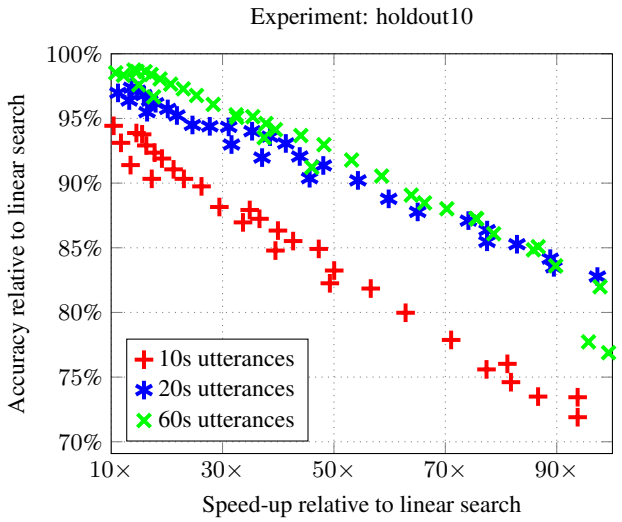

Fig. 2. Speedup vs. accuracy trade-off for the holdout10 experiments. Each point corresponds to a choice of parameters for our LSH data structure. We vary k (the number of hyperplanes) from 6 to 20 in steps of 2 and l (the number of hash tables) from 80 to 300 in steps of 20.

contrast, the search for matches across videos is more challenging and hence requires iterating over a larger set of candidates.

We also compare our LSH-based retrieval algorithm with kd-trees, another popular method for nearest neighbor search in high dimensions [27]. In particular, we use the ANN library [28]. While kd-trees show superior performance on the simple 10tests data set, LSH is significantly faster on the more realistic holdout10 data set. Again, we suppose that this difference is due to the more challenging geometry of searching across videos.

Note that our results are not directly comparable with the speedups reported in [21]. In addition to using LSH, the authors also accelerate the computation of their utterance comparison model with a kernel function and include the resulting performance improvement in their LSH speedup. In particular, the authors report a speedup of 911× for a linear search using their kernel function, compared to a full evaluation of their utterance comparison model as baseline. Expressed in our metrics (i.e., after subtracting the time spent on computing the vector representation), their LSH scheme achieves a speedup of about 12× for *matched* conditions.

5. CONCLUSION

We have proposed a fast retrieval method for speaker identification in large data sets. Our work is based on combining two powerful approaches that interact via the cosine distance: locality sensitive hashing, which enables fast nearest neighbor search, and i-vectors, which provide good identification accuracy. Results on a realistic, large data set from YouTube show that we can achieve speedups of one to two orders of magnitude while sacrificing only a very small fraction of the identification accuracy. Hence our approach is a very promising candidate for large-scale speaker identification. Moreover, LSH could also be very useful for other large-scale applications of i-vectors, such as clustering.

6. ACKNOWLEDGEMENTS

L.S. would like to thank Piotr Indyk for helpful discussions.

	Experiment					
	10tests			holdout10		
	10s	20s	60s	10s	20s	60s
Database size	44,348	44,394	22,213	51,736	51,803	31,104
Query i-vectors	1,200	1,200	1,200	740	739	740
Baseline accuracy	99.0%	99.5%	99.7%	53.4%	64.4%	74.2%
Baseline time (ms)	6.65	6.66	3.38	7.70	7.76	4.67
LSH accuracy	91.0%	95.6%	98.1%	50.1%	60.6%	70.6%
LSH time (ms)	0.071	0.045	0.023	0.495	0.220	0.132
Relative accuracy	**92.0%**	**96.1%**	**98.4%**	**93.8%**	**94.0%**	**95.1%**
Relative speedup	**93×**	**149×**	**148×**	**16×**	**35×**	**35×**
Hyperplanes (k)	18	18	18	14	16	16
Hash tables (l)	220	120	80	280	280	280
kd-tree rel. accuracy	95.2%	96.4%	98.4%	92.9%	93.7%	95.4%
kd-tree rel. speedup	568×	987×	773×	3.6×	2.7×	3.0×

Table 1. Summary of the data sets and results for the two types of experiments (10tests, holdout10) and three utterance lengths (10s, 20s, 60s). The relative accuracy is the ratio (LSH accuracy)/(baseline accuracy) and the relative speedup is the ratio (baseline time)/(LSH time). For each experiment, we present a choice of parameters for our LSH data structure so that we achieve a relative accuracy of at least 90%. The relative accuracy and speedup for kd-trees is also reported relative to the baseline.

7. REFERENCES

[1] D.A. Reynolds, "An overview of automatic speaker recognition technology," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2002.

[2] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *ACM Symposium on Theory of Computing (STOC)*, 1998.

[3] S. Har-Peled, P. Indyk, and R. Motwani, "Approximate nearest neighbor: Towards removing the curse of dimensionality," *Theory of Computing*, vol. 8, no. 14, pp. 321–350, 2012.

[4] "Youtube statistics," `http://www.youtube.com/yt/press/statistics.html`, accessed 3 November 2013.

[5] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey, "Streaming similarity search over one billion tweets using parallel locality sensitive hashing," in *International Conference on Very Large Data Bases (VLDB)*, 2014.

[6] P. Kenny, "Joint factor analysis of speaker and session variability: theory and algorithms," Tech report CRIM-06/08-13, 2005.

[7] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2011.

[8] A.O. Hatch, S.S. Kajarekar, and A. Stolcke, "Within-class covariance normalization for SVM-based speaker recognition.," in *International Conference on Spoken Language Processing (INTERSPEECH)*, 2006.

[9] P. Kenny, "Bayesian speaker verification with heavy-tailed priors," in *Odyssey: The Speaker and Language Recognition Workshop*, 2010.

[10] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

[11] A.Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of Sequences*, 1997.

[12] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *ACM Symposium on Theory of Computing (STOC)*, 2002.

[13] S. Baluja and M. Covell, "Content fingerprinting using wavelets," in *European Conference on Visual Media Production (CVMP)*, 2006.

[14] M. Magas, M. Casey, and C. Rhodes, "mHashup: fast visual music discovery via locality sensitive hashing," in *ACM SIGGRAPH new tech demos*, 2008.

[15] M. Casey, "AudioDB: Scalable approximate nearest-neighbor search with automatic radius-bounded indexing," *The Journal of the Acoustical Society of America*, vol. 124, no. 4, pp. 2571–2571, 2008.

[16] M. Casey and M. Slaney, "Fast recognition of remixed music audio," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2007.

[17] M. Casey and M. Slaney, "Song intersection by approximate nearest neighbor search," in *International Conference on Music Information Retrieval (ISMIR)*, 2006.

[18] M. Casey, C. Rhodes, and M. Slaney, "Analysis of minimum distances in high-dimensional musical spaces," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 5, pp. 1015–1028, 2008.

[19] M.A. Pathak and B. Raj, "Privacy-preserving speaker verification as password matching," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[20] J.P. Campbell Jr., "Testing with the YOHO CD-ROM voice verification corpus," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1995.

[21] W. Jeon and Y. Cheng, "Efficient speaker search over large populations using kernelized locality-sensitive hashing," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[22] W. Jeon, C. Ma, and D. Macho, "An utterance comparison model for speaker clustering using factor analysis," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011.

[23] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[24] P. Kenny, P. Ouellet, N. Dehak, V. Gupta, and P. Dumouchel, "A study of interspeaker variability in speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 5, pp. 980–988, 2008.

[25] A. Andoni and P. Indyk, "Efficient algorithms for substring near neighbor problem," in *ACM-SIAM symposium on Discrete algorithm (SODA)*, 2006.

[26] "Youtube search for google tech talks," `https://www.youtube.com/results?q=GoogleTechTalks`, accessed 3 November 2013.

[27] J. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, Sept. 1975.

[28] D. Mount and S. Arya, "Ann: A library for approximate nearest neighbor searching," `http://www.cs.umd.edu/~mount/ANN/`, accessed 3 November 2013.