Random Projections for Large-Scale Speaker Search

Ryan Leary, Walter Andrews

Raytheon BBN Technologies

ryan@bbn.com, wandrews@bbn.com

Abstract

This paper describes a system for indexing acoustic feature vectors for large-scale speaker search using random projections. Given one or more target feature vectors, large-scale speaker search enables returning similar vectors (in a nearest-neighbors fashion) in sublinear time. The speaker feature space is comprised of i-vectors, derived from Gaussian Mixture Model supervectors. The index and search algorithm is derived from locality sensitive hashing with novel approaches in neighboring bin approximation for improving the miss rate at specified false alarm thresholds. The distance metric for determining the similarity between vectors is the cosine distance. This approach significantly reduced the search space by 70% with minimal increase in miss rate. When combined with further dimensionality reduction, a reduction of the search space by over 90% is also possible. All experiments are based on the NIST SRE 2010 evaluation.

Index Terms: speaker identification, i-vectors, locality sensitive hashing, indexing, search

10.21437/Interspeech.2014-14

1. Introduction

The recent advancements in speaker recognition systems have increased the utility and desirability of large-scale speaker search across many fields, including government, law enforcement, and private industry. Building on improved speaker recognition systems, an efficient large-scale query mechanism would enable both standing queries (i.e. find a known speaker) as well as ad-hoc, on-demand queries (i.e. find more examples of similar speakers).

The speaker recognition system proposed by Dehak [1] projects the Gaussian Mixture Model (GMM) supervector space of speech utterances into a single low-dimensional vector (i-vector) suitable for speaker, language and gender recognition. Further work demonstrated the effectiveness of cosine similarity for final decision scoring [2]. Ad-hoc speaker search by example, then, can be performed by computing the nearest neighbors of the query vector. While i-vectors are low-dimensional in the context of GMM supervectors (hundreds of components instead of tens of thousands), there is no known general-purpose exact solution for nearest-neighbor search of vectors with similar dimensionality that is both polynomial in preprocessing time and polylogarithmic in search time. Near-real time example search of i-vectors enables researchers to generate new or revise existing models without waiting to rescore an existing corpus.

This work intends to demonstrate an approximate technique for efficiently searching a large, high-dimensional dataset in sublinear time while supporting volume indexing in real time. Sublinear search time is achieved by probabilistically pruning vectors which are unlikely to match the query vector. The system requires no domain-specific data for training and supports polynomial-time indexing. Particularly, we use locality-sensitive hashing based on random projection to group vectors into bins with similar vectors. Additional vectors can be added later without changing the existing hashing. At query time, a number of probable bins close to the query vector are hypothesized. All vectors in the hypothesized bins are scored against the query vector according to the cosine distance metric. A number of parameters are available for tuning the system according to desired memory usage, search time, and search accuracy.

The proposed system demonstrates indexing and search of i-vectors scaling to tens or hundreds of millions of records on a small cluster comprised of commodity hardware. A massive reduction in the amount of data searched may be achieved with no impact to false alarm rate and modest increases in miss rate when compared to an exhaustive nearest-neighbor search.

2. Speaker Indexing and Search

The cosine distance classifier, described in section 2.2, enables additional flexibility over earlier speaker recognition schemes (i.e. SVM- or GMM-based). An i-vector model of a speaker is represented by the mean of any i-vectors extracted for a given speaker. Search is therefore possible with as little as a single example i-vector. Two different search architectures may be considered for large-scale speaker search using i-vectors.

One such option is the "continuous query" or "speaker spotting" scheme that is common with earlier speaker recognition systems. This scheme requires a priori knowledge of target, or *known*, speakers. Only scores between new vectors and known speakers are indexed.

A more flexible option is to perform queries by scoring vectors on-demand. In this scheme, the cosine distance is calculated between a target i-vector (model) and all indexed i-vectors when requested. In this method, no model-specific computations are done when vectors are indexed, and since no models are predefined, an index per model does not need to be maintained. As a result, in an on-demand system, the search procedure for existing models is the same procedure used for new or updated models (and, by extension, for known and unknown speakers). A search requires returning the nearest neighbors of the query vector, v_{target}, as well as the similarity (or distance) scores.

2.1. Baseline System

I-vectors are sufficiently high-dimensional to be subject to the "curse of dimensionality" Indexing or similarity search of n vectors with polynomial preprocessing in $\Re^d$ for large d has been shown empirically and theoretically to degrade to search time linear in n and d [3]. Therefore, a linear search system is built to provide baseline results. When provided a query vector, the cosine distance is calculated between the query vector and all vectors in the system. The baseline system is optimal in search accuracy and serves as a benchmark.

2.2. Scoring

Work by Dehak, et. al [2] demonstrated the effectiveness of cosine distance computed between the target speaker model, v_{target}, and a test vector, v_{test} as the decision score. This method is advantageous in a large speaker search system for several reasons. Most notably, unlike using a SVM with a cosine kernel or in Joint Factor Analysis (JFA), no speaker enrollment is necessary. Speaker models may be created by computing a weighted average i-vector based on i-vectors extracted from examples of the same speaker.

$$\text{score}(v_{\text{target}}, v_{\text{test}}) = \frac{\langle v_{\text{target}}, v_{\text{test}} \rangle}{\|v_{\text{target}}\| \|v_{\text{test}}\|} \tag{1}$$

Accept/reject decisions are made by comparing the score calculated in equation 1 to an experimentally determined threshold, t. The performance of the system may be tuned with adequate development data to bias toward false alarms or false negatives. Scores exceeding the threshold are considered matches. As t increases, the likelihood of misses increases while simultaneously reducing the likelihood of false alarms.

3. Locality-Sensitive Hashing

For sublinear search complexity, i-vectors unlikely to exceed the established threshold when scored should be pruned and ignored at search time.

Locality-sensitive hashing is one such technique which groups vectors into 'bins' based on some distance metric operating on the vectors. When hashed, vectors that are close to each other in the original space are mapped into the same bin with high probability [4]. This hashing operation effectively clusters the input data into a large number of very small clusters. The typical search use case is to hash the query vector, determine the bin it hashes to, and compute the distance between the query vector and all vectors previously hashed to that bin.

Hash amplification is used to increase the number of bins a vector may be hashed to [5]. Amplification is a means of combining multiple, independent, hash functions drawn from the same family to create a new family with different probability bounds. In practice, we use a combined `and-or` construction where `and` increases the number of bins vectors are distributed across, while `or` increases the number of bins searched, thus reducing the risk that a given hyperplane will split a cluster of vectors.

3.1. Random Projections

Input i-vectors are projected against random hyperplanes to a lower-dimensional subspace according to the Johnson-Lindenstrauss lemma, which states

> If points in a vector space are projected onto a randomly selected subspace of suitably high dimension, then the distances between the points are approximately preserved.

For cosine distance, projecting against a single random hyperplane drawn from a d-dimensional Gaussian distribution [6] splits the data into two bins (the number of possible bins is equal to 2^k where k is the number of random hyperplanes) and is $\left(d_1, d_2, \frac{(180-d_1)}{d_1}, \frac{d_2}{180}\right)$ -sensitive when d_1 and d_2 are given in degrees.

To satisfy the `and` construction, a random k x d matrix R is drawn from a standard normal gaussian distribution, where d is the original dimensionality of the i-vectors to be searched, and k is the desired number of hyperplanes. We choose $k << d$.

An original i-vector, $\vec{v}$, is first projected into the lower dimensional subspace, producing an intermediate vector, $v_{\vec{\text{RP}}}$.

$$v_{\vec{\text{RP}}} = R\vec{v} \tag{2}$$

We define a sign function:

$$\text{sgn}(x) = \begin{cases} 1 & : x > 0 \\ 0 & : x \leq 0 \end{cases} \tag{3}$$

and binning function for arbitrary vector $\vec{x}$

$$h(\vec{x}) = \sum_{i=1}^{k} \text{sgn}(x_i) \cdot 2^{(k-i)} \tag{4}$$

such that the bin, z_0, the original i-vector $\vec{v}$ is hashed into is calculated as

$$z_0 = h\left(v_{\vec{\text{RP}}}\right) \tag{5}$$

The binning function in equation 4 emits an integer-valued bin id z, $0 \leq z < 2^k$. Each i-vector requires only one bit per hyperplane used in the random matrix to store (k bits).

To satisfy the `or` construction, multiple random matrices, R_b, are chosen and this process is repeated. Each vector is hashed according to each of the b random matrices, and a database index which maps bin value to vectors in that bin is created. At search time, the query vector is hashed according to equation 5 and each random matrix, returning b bin ids (one per hash function) whose vectors are candidates for scoring. The database is subsequently queried for all vectors that match the bins calculated, and the union of returned vectors are scored against the query vector as in the baseline system.

The `or` construction is necessary to improve the accuracy of the search because of the marked decrease in recall probability caused by the `and` construction. However, for each additional random matrix that is projected against, an additional database index is required to provide fast lookups. The size of b is thus constrained by hardware due to the index size growing linearly in b to $\text{O}(bn)$.

The searching system is used only for pruning and therefore does not increase the false alarm rate over a baseline system.

3.2. Neighboring Bin Approximation

Using the random projection approach outlined in the previous section, a locality-sensitive hash based search system can be built [7]. To increase scalability, we aim to relax the constraints slightly by introducing a notion of neighboring bins – thereby reducing the number of indices, b, required.

Neighboring bins are alternative hypotheses for which bin similar vectors may have hashed to based on the assumption that the data for any particular speaker is structured and should fall in a small (yet greater than one) number of bins. At query time, in addition to the bin the query vector would be hashed to, alternative bins near the vector are hypothesized. Vectors hashed into the hypothesized bins are also returned as part of the search process and scored.

3.2.1. Hamming

In the projected subspace, hamming distance approximates cosine distance as the number of random hyperplanes, k, is increased [8]

$$\cos(q, v) \approx \cos\left(\frac{H\left(h(Rq), h(Rv)\right)}{k}\pi\right) \tag{6}$$

where $H(\cdot,\cdot)$ denotes Hamming distance, and h is as defined in equation 5.

Taking this into consideration, it is possible to hash a query vector and then return vectors from all bins within a certain hamming distance. Unfortunately, lexicographic sorting of bin ids does not necessarily cluster neighbors. The authors in [9] devised a randomized hashing algorithm. When new vectors are indexed, they are hashed according to equation 5 in section 3.1. The bits representing the bin integer value are shuffled according to P permutations specified earlier and inserted into a unique index for each permutation. At search time, the query bin is calculated as described for indexing, then permuted, with the vectors in B neighboring bins (lexicographically) scored and returned.

If implemented as described above, the number of indexes will increase from b (one per hash function) to Pb (P indexed shufflings per hash function). This is not strictly necessary, as it is possible to perform the bit permutation, iterate over the neighbors, and on each iteration, compute the inverse permutation to obtain a bin id in the original projected space. The primary drawback to this approach is the lack of adequate information to determine high-likelihood neighboring bins.

3.2.2. Query-directed

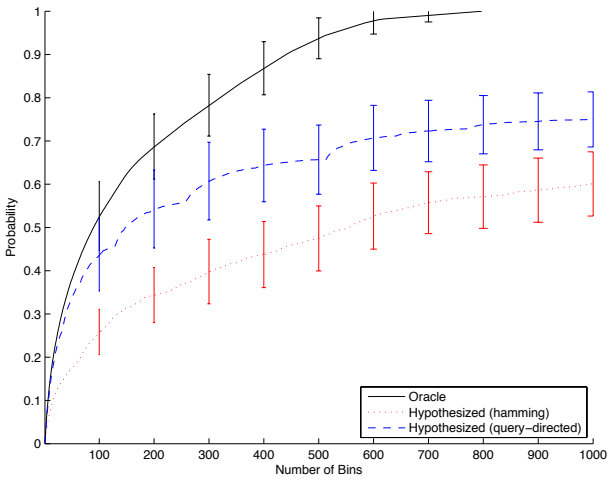

Figure 1: *CDF of 1000 synthesized examples of a speaker drawn from SRE10 dev set within-class covariance (average of ten trials),* $k = 16$

We build on the Hamming distance-based neighboring bin approximation method to choose higher likelihood bins to return and score. The query i-vector (prior to being hashed) contains valuable information that can be used to determine which hyperplane boundaries the vector will be projected closest to after hashing. Examples of a given speaker should all fall within a short distance of each other, but there is no guarantee that the short distance does not span a hyperplane boundary. Therefore, it is possible to find examples similar to the query vector by searching neighboring bins intelligently.

For a given query vector Q, we project to the lower dimensional subspace according to equation 2 and temporarily store as Q_{RP}. The bin, z_0, the query vector, Q, would hash to is calculated as per equation 5. We then sort the element-wise absolute value of the projected vector and store the indices of the sort order in S as follows:

$$S = \text{argsort}(|Q_{RP}|) \tag{7}$$

Similarly to the hamming approach in section 3.2.1, we choose a number, l, of neighboring bins from which to calculate and score constituent vectors. The l neighboring bins are determined by iterating from 1 to l and calculating equations 8 and 9. On the i'th iteration, we compute an integer that when XOR'd with q is a neighboring bin in hamming-space.

For the ith neighboring bin, a mask, p_i, indicating which bits to switch in q_0, is calculated by

$$p_i = \sum_{j=1}^{l} \text{sgn}(i \wedge 2^{(j-1)}) \cdot 2^{k-S_j} \tag{8}$$

(Note: '$\wedge$' indicates a bitwise-AND and '$\oplus$' indicates a bitwise-OR). Finally, the ith-closest neighboring bin is calculated

$$z_i = z_0 \oplus p_i \tag{9}$$

Equation 8 uses S to generate a mask to negate bits in the integer representation of q_0, similarly to perturbation vectors in [10]. Since each bit is essentially a quantized dot product of the query vector and test vector, changing bits that were (in Q_{RP}) closest to zero may be prioritized. It is assumed that the closer a query vector falls to a hyperplane, the more likely it is that similar vectors will be hashed into bin(s) adjacent to that hyperplane.

Figure 1 illustrates the above methods for approximating neighboring bins. We test the above methods by generating and hashing 1000 synthetic i-vectors representing a single speaker. The 'ideal' series, generated using truth data, returns bins in sorted order by number of target cuts in each bin. The query-directed approach, while underperforming compared to the series generated with truth data, significantly outperforms the hamming-based approach, requiring nearly a third fewer bins be analyzed to return half of the indexed i-vectors. Neither the hamming nor query-directed approaches return all i-vectors without performing an exhaustive search.

4. Experimental Setup

The query-directed random projection system has three parameters that may be tuned to tailor system performance: k, the number of hyperplanes per hash function to project against, b, the number of random hash functions to use (and therefore database indexes to create), and l, the number of neighboring bins to search in each hash function.

By tuning these three parameters, we seek to maximize the search accuracy (as defined in section 4.2) while minimizing the search time complexity and also being space efficient in the database.

Indexing random vectors into the lower-dimensional subspace empirically results in vectors being nearly uniformly distributed among bins. In a system with even tens of millions of indexed vectors, the distribution of vectors among bins will be quite sparse. Two configurations of k (16, 30) are presented in these results. Choosing $k > 32$ would require, in a sparse representation, that bin ids be represented with 64-bit integers instead of 32-bit integers, doubling the memory requirement. Finally, while calculating the nearest bins is trivial, computationally, it is exponential in k, so there is an overhead tradeoff that must be considered. The overhead penalty can be amortized over the time spent calculating, and thus a 'large' k (≤ 32) is not detrimental to search performance for large databases.

		400-dim. i-vectors, $k = 30$			150-dim. i-vectors, $k = 30$		
v_s	b	EER	minDCF[1]	Recall[2]	EER	minDCF[1]	Recall[2]
1*	0	7.4807	0.7512	0.3056	2.9726	0.4706	0.5691
0.1	10	40.0578	0.7908	0.2603	9.9567	0.4779	0.5619
0.05	10	53.4198	0.8263	0.2248	17.1716	0.4929	0.5469
0.05	5	60.7216	0.8553	0.1844	24.5309	0.5275	0.5123
0.01	10	78.8745	0.9140	0.0860	39.0476	0.5820	0.4577

Table 1: *Results of selected search system tunings with 400-dimensional and 150-dimensional LDA-WCCN i-vectors.* $v_s \approx \frac{lb}{2^k}$ *is the approximate fraction of i-vectors returned and scored, and* b *is the number of hash functions and indices used. The first row is the baseline, exhaustive system provided for comparison. minDCF[1] is the norm-minDCF. Recall[2] is the recall at the minDCF operating point.*

		150-dim. Search Test, $k = 16$		
v_s	b	Recall	Overhead	Total
1*	0	0.6346	–	1.4206s
0.1	10	0.5989	0.0102s	0.1970s
0.05	10	0.5544	0.0076s	0.1304s
0.05	5	0.5180	0.0053s	0.1115s
0.01	10	0.3567	0.0033s	0.0335s

Table 2: *Average per-query speed of* 1800 *queries over a corpus of* 35163 *vectors. Overhead is the time spent calculating candidate vectors based on nearest bins. It is exponential in* k *and linear in* v_s*. Recall is calculated on targets found in top* 15 *results.*

4.1. Data

Experiments in this work are performed on i-vectors extracted from data provided in the NIST SRE 2010 evaluation [11] and trained according to the core training condition. All training samples are two-channel telephone conversational excerpts of approximately five minutes total duration. Experiments are performed on the male-only examples in common evaluation condition 5, which specifies normal vocal effort conversational telephone speech in training and test.

The i-vector extractor is configured to extract 400-dimensional i-vectors via a 40-dimensional feature, diagonal covariance, 1024-mixture Gaussian universal background model. Further, the i-vectors in all cases are centered, whitened, and scaled to fall on the unit sphere. In some cases, the i-vectors are further reduced using Linear Discriminant Analysis and Within-class Covariance Normalization (LDA-WCCN). These results are presented separately.

A 'baseline' result indicates scoring using no random projections (exhaustive linear search). For each query, and specified b and l values, hypothesized bins for each hash function are calculated. Test vectors are hashed according to equation 5 for each of the b random matrices. If any of the hypothesized bins for the query vector match the actual bins for the test vector, the cosine distance score is computed as normal. If the test vector's bins are not hypothesized, it is scored as -1 (miss).

4.2. Metrics

Speaker recognition systems are subject to two types of errors, missed detection and false alarms. Large scale speaker search systems are also represented by the same errors.

The minimum Decision Cost Function (minDCF) is the preferred measure when accounting for application specific costs and priors. The DCF is defined as

$$\begin{aligned}\mathrm{DCF}_\theta = {} & \mathrm{C}_{\text{Miss}} \cdot \mathrm{P}_{\text{Miss}|\theta} \cdot \mathrm{P}_{\text{Speaker}} \\ & + \mathrm{C}_{\text{False Alarm}} \cdot \mathrm{P}_{\text{False Alarm}|\theta} \cdot \mathrm{P}_{\text{Impostor}} \end{aligned} \quad (10)$$

where C_{Miss} is cost of a miss, $\mathrm{P}_{\text{Speaker}}$ is the prior probability of true speaker attempt, $\mathrm{C}_{\text{False Alarm}}$ is the cost of a false alarm, and $\mathrm{P}_{\text{Impostor}} = 1 - \mathrm{P}_{\text{Speaker}}$ is the prior probability of an impostor attempt. The values of C_{Miss}, $\mathrm{C}_{\text{False Alarm}}$, and $\mathrm{P}_{\text{Speaker}}$ are set to the primary evaluation metric defined in [11].

Large scale search tasks often emphasize low false alarm rates. A large number of high-scoring false alarms makes it difficult to find relevant conversations. As such, we focus our results on performance when the system is tuned to operate at the minimum DCF point.

5. Results

The search algorithm was applied to both raw and LDA-WCCN i-vectors extracted from SRE 2010 data. A subset of configurations of search systems tested is included in Table 1. As seen in the table, the search system performs dramatically better when used with LDA-WCCN i-vectors. This can be attributed to the search system's sensitivity to within-class covariance.

The baseline and LDA-WCCN baseline systems achieved recall rates of 0.3056 and 0.5691, respectively, at the minDCF point. In one configuration, 90% of the data in the corpus is pruned as non-targets and achieved recall rates of 0.2603 and 0.5619. With the LDA-WCCN system, an additional order of magnitude less data may be searched while decreasing recall 19.6% to 0.4577.

Recall rates may be improved by relying more on additional indices (as described in section 3.1) than on returning vectors based the query-directed neighboring bins. This is evidenced in the two trials where $v_s = 0.05$. The recall is 3-4 points higher in both systems when $b = 10$. It is important to note the required RAM increases linearly as a function of number of vectors, n, and hash functions/indices b.

If we constrain our system to operate on no more than 2^{32} vectors with $k \leq 32$, we can generate 64-bit tuples mapping a bin id to vector id. Ignoring overhead, each tuple can be stored in 8 bytes per vector per hash function, therefore one index for one million vectors requires 7.63 MiB, while an index for one billion vectors requires ~7.45 GiB: feasible on commodity hardware. Once the neighboring bins are hypothesized for each index, a small distributed cluster can be searched, reducing the memory requirement on any given machine to tens of gigabytes.

The timing data in Table 2 was measured using a single-threaded Python implementation running on a workstation with a 2 GHz processor and 8 GB of RAM. The 'total' time may be reduced in an optimized implementation by scoring and identifying additional candidate vectors in parallel.

6. Conclusion

We have presented a solution to approximate nearest-neighbor search in high dimensions specifically tailored to speaker recognition using i-vectors. The search system is capable of 1-2 orders of magnitude speedup over an exhaustive search depending on desired search accuracy. We found that a query-directed traversal of the hamming space is possible and outperforms a randomized traversal as suggested in other works. Further improvements to the query-directed approach are possible by tailoring traversal depth on a per-index basis depending on query vector and will be explored in a future work.

7. References

[1] N. Dehak, P. Kenny, R. Dehak, O. Glembek, P. Dumouchel, L. Burget, V. Hubeika, and F. Castaldo, "Support vector machines and joint factor analysis for speaker verification," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, 2009, pp. 4237–4240.

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

[3] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proceedings of the 24rd International Conference on Very Large Data Bases*, ser. VLDB '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 194–205. [Online]. Available: http://dl.acm.org/citation.cfm?id=645924.671192

[4] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 518–529. [Online]. Available: http://dl.acm.org/citation.cfm?id=645925.671516

[5] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011.

[6] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *In Proc. of 34th STOC*. ACM, 2002, pp. 380–388.

[7] L. Schmidt, M. Sharifi, and L. Moreno, "Large-scale speaker identification," in *Acoustics, Speech and Signal Processing, 2014. ICASSP 2014. IEEE International Conference on*, 2014.

[8] A. Jansen and B. Van Durme, "Efficient spoken term discovery using randomized algorithms," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, 2011, pp. 401–406.

[9] A. Jansen and B. V. Durme, "Indexing raw acoustic features for scalable zero resource search," in *Proc. Interspeech*, 2012.

[10] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*, ser. VLDB '07. VLDB Endowment, 2007, pp. 950–961. [Online]. Available: http://dl.acm.org/citation.cfm?id=1325851.1325958

[11] National Institute of Standards and Technology, "The nist year 2010 speaker recognition evaluation plan." [Online]. Available: http://www.itl.nist.gov/iad/mig//tests/sre/2010/NIST_SRE10_evalplan.r6.pdf