INTERSPEECH 2016
September 8–12, 2016, San Francisco, USA

Speaker Linking and Applications using Non-parametric Hashing Methods[†]

Douglas Sturim and William M. Campbell

MIT Lincoln Laboratory, Lexington, MA, USA
{sturim,wcampbell}@ll.mit.edu

Abstract

Large unstructured audio data sets have become ubiquitous and present a challenge for organization and search. One logical approach for structuring data is to find common speakers and link occurrences across different recordings. Prior approaches to this problem have focused on basic methodology for the linking task. In this paper, we introduce a novel trainable non-parametric hashing method for indexing large speaker recording data sets. This approach leads to tunable computational complexity methods for speaker linking. We focus on a scalable clustering method based on hashing—canopy-clustering. We apply this method to a large corpus of speaker recordings, demonstrate performance tradeoffs, and compare to other hashing methods.

Index Terms: speaker recognition, clustering, hashing, locality sensitive hashing.

1. Introduction

We assume that a large corpus of audio recordings with re-occurring speakers is given. Our goal is to structure this corpus in two ways. First, we want to explore methods for quickly performing speaker query-by-example (QBE). I.e., given a recording from a speaker, find all recordings by the same speaker in our corpus. Second, given a QBE method, how can we perform speaker clustering—each clustering should be a single speaker, and a cluster should contain all recordings from that speaker in the corpus. The result of these two steps is a structured organization of the corpus by speaker—we can quickly find the same speaker in multiple recordings.

Two critical tools for speaker linking in large corpora are speaker diarization and speaker embedding. For the first part, since we want to focus on the speaker linking aspect, we assume that diarization has been performed and each recording in our corpus contains only a single speaker. For speaker embedding, we convert our recordings to a (single) speaker vector. This process can be performed with many approaches [1, 2]. For this paper, we focus on using i-vectors [2], but the methods apply to any embedding.

For the task of speaker QBE and recognition, multiple methods have been proposed. First, the most straightforward method [3] is—given a query vector, $\mathbf{x}_q$, perform inner products with all vectors in the corpus, $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. This approach is $O(n)$ computation and $O(n)$ storage, so it grows linearly with the corpus size. A second approach, graph-based query by example uses a graph structure along with random walks to perform retrieval [4]. This approach is computationally $O(1)$ for retrieval, and requires $O(n)$ for storage. A drawback of the method is that it requires an $O(n^2)$ computation for setting up the retrieval data structure. A third approach for QBE is to use (locality-sensitive) hash function methods. Speaker vectors are first converted to binary form via a hash function, and then retrieval is performed using standard computer science hash table methods. Multiple authors have explored this approach [5, 6, 7]. The advantage of this approach is that storage is $O(n)$ and average retrieval complexity is $O(1)$.

For this paper, we use the hash function approach to speaker QBE. Prior approaches have used locality sensitive hashing via random projection [5, 6]. In this paper, we look at non-parametric trainable methods to hashing. The goal is to create data-specific hashing functions with better performance. Non-parametric methods offer the advantage of being a consistent estimator of the underlying probability density function of the data without any prior knowledge [8, 9]. Note that for this paper, we focus on speaker QBE for speaker clustering by using hash functions only and no graph structure. But, speaker QBE with hash functions can be combined with random walks by retrieving a local k-hop graph in the same manner as [4].

Our speaker application combines speaker QBE with linking and clustering. Prior methods for linking and clustering speakers have been studied in multiple contexts including speaker content graphs [10], as a corpus level linking problem [11], as a diarization problem [12], and as a graph clustering problem [13]. The basic approach is to find speaker links—i.e., are two recordings from the same speaker. Putting this all together gives a graph where nodes are recordings and weighted edges represent the confidence of the links. Clustering can be considered an extension of this process where the links must be transitively consistent. I.e., given that recording A and B are the same speaker, B and C are the same speaker, then A and C are the same speaker. Another way to say this is that the speaker content graph is a union of cliques [14].

Our approach to speaker linking and clustering consists of three parts. First, we convert all recordings in the data set to speaker vectors and then use a hash function to convert the data to a binary representation. Second, we index this binary representation using hash tables. Third, we use multiple retrievals to calculate a subset of the full distance matrix to reduce computation by a tunable amount. This approach is inspired by canopy clustering [15]. The resulting speaker method can be tuned both computationally and storage-wise to achieve different levels of performance.

The outline of the paper is as follows. In Section 2, we cover speaker QBE using hashing methods. We review the standard random projection approaches as well as introducing our non-parametric approaches. In Section 3, we discuss clustering and detail the combination of hashing and canopy clustering.

[†]This work was sponsored by ASD Research and Engineering's Rapid Reaction Technology Office under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

http://dx.doi.org/10.21437/Interspeech.2016-468

Finally, in Section 4, we apply our methods to a large NIST speaker corpus and detail experimental results, trade offs, and performance.

2. Hashing Techniques

Hashing is a common technique for finding nearest neighbors of a given vector, $\mathbf{x}$. A hashing function is defined,

$$h(\mathbf{x}) : \mathbb{R}^d \longrightarrow B^l \tag{1}$$

where $B = \{0, 1\}$ and l is the number of bits. The hashing function should have the property that if $\mathbf{x}$ and $\mathbf{y}$ are close, their corresponding hash values $h(\mathbf{x})$ and $h(\mathbf{y})$ have close Hamming distance.

A common technique used in hashing is to define m randomly selected hashing functions h_i from a family of hash functions all with the same number of output bits l. Given a set of vectors $\mathbf{X} = \{\mathbf{x}_i\}$, $i = 1, \dots, n$, the data are encoded using all of the hash functions, $\mathbf{h}_i(\mathbf{x})$. A retrieval function,

$$\begin{aligned} \mathcal{H}_i(b) &: B^l \longrightarrow \mathcal{P}(\{1, \dots, n\}) \\ \mathcal{H}_i(b) &= \{j | h_i(\mathbf{x}_j) = b\} \end{aligned} \tag{2}$$

maps bits to the set of indices of the data; $\mathcal{P}(\cdot)$ denotes the power set. Multiple functions can be used to ensure that a close neighbor vector will be eventually retrieved. The retrieval can be implemented with $O(1)$ average search complexity using an inverted index. More details on multiple hash functions and implementation can be found in [16].

2.1. Locality Sensitive Hashing (LSH)

A standard baseline method for vector-based hashing is Locality Sensitive Hashing (LSH) [17, 18]. LSH provides probabilistic bounds for near items having the same hash value. A typical method for implementing LSH for vectors is to use random projection.

In more detail, assume that we want to encode a unit norm vector $\mathbf{x}$. A random matrix, M_i, with l columns $\mathbf{M}_{i,j}$ of dimension d is generated. Then hash values for the ith hash function, $h_i()$, are generated by,

$$h_{i,j}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{M}_{i,j}^t \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where j indicates the jth bit of the output. Intuitively, each bit represents the side of the hyperplane where the vector lies.

2.2. Distance Based Hashing (DBH)

An alternate to LSH is distance-based hashing (DBH). Our implementation closely follows the original algorithm [19] with the exception of compute performance optimizations. Distance based hashing was chosen for comparison because of its similarities with other non-parametric methods.

As with non-parametric hashing presented in the next section, distance-based hashing is initialized by selecting a set of $2l$ seed vectors that will serve as a basis for a hashing function to encode a target vector. These seed vectors should be chosen from a set of data that will be representative of the target data to be encoded.

The base function used for hashing is the pseudo-line-projection proposed in [19],

$$d_{\mathbf{x}_1,\mathbf{x}_2}(\mathbf{x}) = \frac{d(\mathbf{x},\mathbf{x}_1)^2 + d(\mathbf{x}_1,\mathbf{x}_2)^2 - d(\mathbf{x},\mathbf{x}_2)^2}{d(\mathbf{x}_1,\mathbf{x}_2)^2} \tag{4}$$

Algorithm 1 Non-Parametric Hashing (NPH)

```
Input: Speaker vector, $\mathbf{x}$
Outputs: Encoded vector, $h_{\mathrm{NPH},i}(\mathbf{x})$, of $l$ bits for $i = 1, \dots, m$
For each hashing function, choose $\mathcal{S}_i = \{\mathbf{s}_{i,1}, \dots, \mathbf{s}_{i,l}\}$ seed speaker vectors from an initialization data set and $\mathcal{T}_i = \{T_{i,1}, \dots, T_{i,l}\}$ tolerance intervals using the $k$-nearest neighbors
for $i = 1, \dots, m$ do
  for $j = 1, \dots, l$ do
    if $d(\mathbf{x}, \mathbf{s}_{i,j}) < T_{i,j}$ then
      $h_{\mathrm{NPH},i,j}(\mathbf{x}) = 1$
    else
      $h_{\mathrm{NPH},i,j}(\mathbf{x}) = 0$
    end if
  end for
end for
```

where $d(\cdot, \cdot)$ is the Euclidean distance between two vectors. The vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are (fixed) seed vectors. Encoding of an input vector is accomplished in a similar manner to LSH in (3) using the pseudo-line-projection (4),

$$h_{\mathrm{DBH},i,j}(\mathbf{x}) = \begin{cases} 1 & \text{if } d_{\mathbf{x}_{i,j,1},\mathbf{x}_{i,j,2}}(\mathbf{x}) \in [t_{i,j,1}, t_{i,j,2}] \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Points are encoded depending on if they fall within or outside an interval. The interval is defined by thresholds $t_{j,1}$ and $t_{j,2}$. In order to form two reference points for the pseudo-line-projection equation, the $2l$ initialization vectors are randomly selected to form l pairs. The bits of the hash are computed by equation (5) over the $2l$ pairs forming an l-bit hash.

2.3. Non-Parametric Hashing (NPH)

Non-parametric pattern recognition/clustering methods have the advantage of being an unbiased representation of the data set modeled. The disadvantage of these techniques is that they are computationally expensive. The construction of our non-parametric hashing (NP-hashing) leverages the computational speed of hashing with the desirable modeling qualities of a non-parametric approach.

The basic encoding approach is shown in Algorithm 1. As with distance-based hashing, the first step of NPH is the selection of a set, $\mathcal{S}$, of l seed vectors. Also, the seed vectors should be chosen from a set of vectors that are closely representative of the target data. Tolerance intervals or hyperspheres are formed by using the $\mathcal{S}$ seed vectors as a set of centers. The set of tolerance intervals, $\mathcal{T}$, is then formed by using the largest distance from the k nearest neighbors to these centers.

The input speaker vectors are encoded into a l-bit hash by comparing against these tolerance intervals. An input speaker is compared against each tolerance interval. If the input falls with the tolerance interval it is coded as a 1. If it lies outside the tolerance intervals it is coded as a 0. In this manner, the l-bit hash is encoded.

3. Clustering Techniques

3.1. Prior Methods

The computation and storage involved in clustering is a major issue for large-scale implementation. The two basic steps involved are—the distance matrix computation and the clustering

Algorithm 2 Canopy Construction Algorithm

```
**Inputs**: Speaker vectors, $\{\mathbf{x}_i\}$, $i = 1, \ldots, n$; hashing retrieval methods, $\mathcal{H}_i(\cdot)$, $i = 1, \ldots, m$; and a similarity threshold $T_s$
**Outputs**: A canopy (set of sets), $\mathcal{C} = \{C_1, \ldots, C_{n_c}\}$
Let $\mathcal{C} = \{\}$, $\mathcal{I} = \{1, \ldots, n\}$, and $i = 1$
**while** $\mathcal{I}$ is not empty **do**
    Pick a random $i_1 \in \mathcal{I}$
    Perform retrievals, $R_j = \mathcal{H}_j(h_j(\mathbf{x}_{i_1}))$ for $j = 1, \ldots, m$
    $C_i = (\cup R_j) \cap \mathcal{I}$; add $C_i$ to $\mathcal{C}$
    Calculate a similarity score for each $p$ in $C_i$, $s(p) = \frac{1}{m}\left|\{j | p \in R_j\}\right|$
    Remove all $p$ from $\mathcal{I}$ with $s(p) \geq T_s$
    $i = i + 1$
**end while**
```

algorithm. In this paper we address the former and consider the latter an area with many choices; see, for example, the many graph-based and standard methods in [13].

A straightforward approach to distance matrix computation is to compute all pairs of distances $d(\mathbf{x}, \mathbf{y})$ and store them in a matrix. This involves $O(n^2)$ storage and $O(n^2)$ flops where n is the number of vectors to cluster. These resource requirements become large quickly; for $100K$ vectors, we require 80 GBs to store the distance matrix.

A step in the right direction is to sparsify the distance matrix by selecting the k closest neighbors. This approach was used with success in [10, 13]. This results in a matrix with storage $O(n)$ for fixed k, but computation requirements are still $O(n^2)$.

To reduce the computation burden, we use the hashing techniques from Section 2 to limit computation. Specifically, our approach is to use hashing to retrieve a candidate set of nearest neighbors and then compute the distance only for those neighbors; we detail this canopy clustering approach more in the next section.

3.2. Canopy Clustering

Canopy clustering consists of three basic parts. First, we construct canopies, $\mathcal{C}$, using hashing functions. Second, distances are constructed based on the canopies. Third, a clustering technique is performed using the computed distances.

The first step, canopy construction, is shown in Algorithm 2. The basic flow of the algorithm is to pick a random member of the vectors to cluster and retrieve everything close using hashing. This process is repeated until the entire set is covered by the resulting canopy, $\mathcal{C}$, which is a set of sets.

The second step of canopy clustering is distance computation. For each canopy, C_i, all the of distances are computed exhaustively in that canopy—i.e., $d(\mathbf{x}_i, \mathbf{x}_j)$ for all i, j in C_i. Note that distinct canopies may have common members so the resulting distance matrix has a block diagonal component with some out-of-block distances computed also.

The third step with canopy clustering is to perform clustering. In this paper, we use standard greedy agglomerative clustering (GAC) with a stopping threshold. Multiple standard link criteria were considered including minimum, maximum, and average. Although GAC is computationally expensive, it is a standard well-performing approach that serves as a baseline. Alternate approaches may have lower computational burden [13]. Another comment on our GAC approach is that the interpretation of sparsity in the distance matrix is non-standard. If a distance in D is not specified, it is assumed to be ∞ not the standard convention of zero.

4. Experiments

4.1. Experimental Setup

The experimental setup was to perform speaker clustering using data from the NIST Speaker Recognition Evaluation (SRE) years 2004, 2006, 2008, 2010 and 2012 [20]. I-vectors were generated with our standard I-vector system [21, 22].

The data was subdivided into training and testing partitions. Table 1 shows the partitioning of the speech corpora used in the experiments. The training data was used for all hyper-parameter training of the I-vector system as well all of the pre-trained parameters of the clustering and hashing functions.

Table 1: Training and testing partitions of the speech corpora

Partition	SRE Years	# of Speech Cuts	# of Speakers
Training	2004, 2005, 2006	17894	2166
Testing	2008, 2010, 2012	18250	1835

4.2. Clustering Experiments

Clustering and hashing both require the setting of pre-trained parameters such as: 1) the number of hashes used, 2) the number of bits used in the hash l, 3) the GAC clustering threshold, and 4) the similarity threshold T_s. To construct a fair clustering experiment, the pre-trained parameters were tuned with the training partition and then applied to the testing data.

Two metrics were used to assess performance of the clustering experiments: 1) adjusted mutual information (AMI) [23], and 2) sparsity of the distance computations. AMI measures the performance of the clustering algorithm. The AMI calculation used is,

$$AMI(U,V) = \frac{MI(U,V) - E\{MI(U,V)\}}{max\{H(U), H(V)\} - E\{MI(U,V)\}} \quad (6)$$

where $MI(U,V)$ is the mutual information between putative clustering set U and ground truth clustering set V. $H(U)$ is defined as the entropy of the set U. Note that an AMI of zero corresponds to chance.

The experiments in this section required multiple experiment sweeping: 1) number of hash bits, 2) number of hash functions, 3) GAC threshold, and 4) similarity clustering threshold. All were conducted by taking random draws of 1000 I-vectors and then conducting clustering experiments over the 1000 vectors. The results were then ensemble averaged.

The sparsity of distance computations is the percentage of distances *not* computed over the entire set to be clustered. Most clustering methods require full matrix of distance computations or $O(n^2)$ computations. This metric evaluates the computational savings of our proposed clustering algorithm.

Figure 1 plots two sets of clustering results using the baseline LSH function. Similar trends are seen for the two other hashing methods, DBH and NPH. The first plot is of adjusted mutual information versus number of bits with a varying number of hash functions. A trend can be seen as the number of bits increases for the hash—the AMI performance decreases as the number of bits increases for a single hash function. This property is due to the fact that the hash becomes too specific and the input points only hash to themselves and not to a locality of

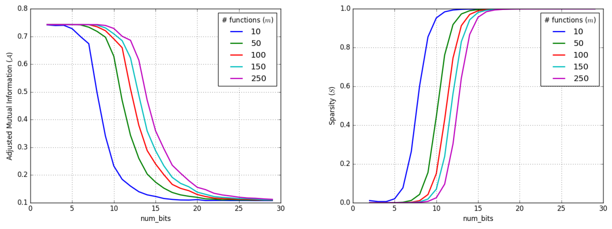

Figure 1: Plots of LSH for adjusted mutual information and sparsity versus number of bits for various number hash functions.

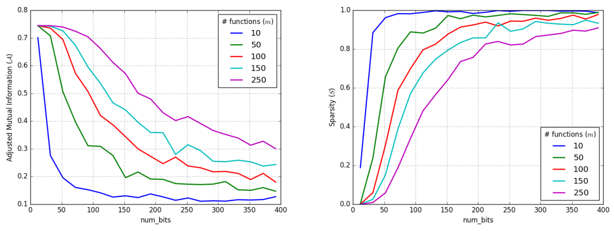

Figure 2: Plots of NPH for adjusted mutual information and sparsity versus number of bits for various number hash functions.

points. This specificity of the retrieval can be controlled by using multiple randomly selected hash function [16]. The retrieval in this case is the union of the retrievals (cf., Algorithm 2).

The second plot of Figure 1 presents the sparsity of distance computations versus the number of bits in the hash. This is also plotted with a varying number of total hash functions. As the number of bits increases the sparsity of distance computations increases. Again this is due to the fact that the hash function is becoming too specific with the increase in the number of bits. The canopy clustering algorithm computes distances over smaller canopies.

Plots of AMI and sparsity versus number of bits for the non-parametric hashing method are shown in Figure 2. Comparing the first plot of Figure 2 with the AMI versus bits of Figure 1, the clustering performance drops off much slower with the increase in the number of hashing bits. Comparing sparsity versus number of bits in Figures 1 and 2, the sparsity of distance computations increases at a slower rate for canopy clustering with non-parametric hashing.

AMI and sparsity of distance computations are a trade-off between clustering performance and computational efficiency. This trade-off can be explored directly by plotting AMI against sparsity. Figure 3 shows two plots for LSH for AMI versus sparsity over a variety of number of hash functions. The first plot is a clustering experiment on the training set of data and the second plot is on the testing set. Better performing systems have curves tending more the upper right of the plot. As expected the system performs better on the training set of data since the systems I-vector hyper-parameters were tuned on the training data. However the clustering system of Figure 3 used clustering parameters greedy agglomerative clustering stopping threshold (GAC) and similarity threshold, that were set to some reasonable settings. A more pragmatic approach would be to set the clustering parameters on the training set of data and then apply the parameters to a cluster experiment.

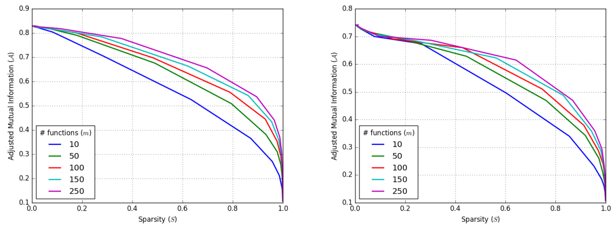

Figure 3: Oracle plots for adjusted mutual information versus sparsity for LSH. The plots are results of the training set and testing set over a varied number of hash functions

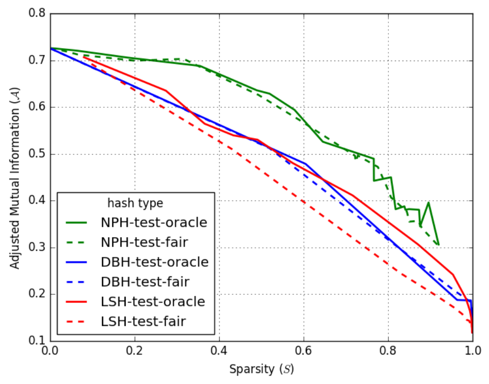

Figure 4: Plot of AMI versus sparsity number hash functions=250

Figure 4 presents results for AMI versus sparsity on the test data set. The solid lines (oracle) are results when the clustering experiment used hyper-parameters trained on test. The dashed lines (fair) are results when the clustering experiment was on the testing data set and clustering parameters are tuned from the training data. Figure 4 shows that the performance drops off slightly in the fair experiments but the parameter tuning is robust. Additionally, the figure shows that the new NPH approach has superior performance to the other methods. Finally, we note that there is still room for significant improvement at high sparsity—further improvements in hashing are possible.

5. Conclusions

In this paper we have introduced a new locality sensitive hashing technique, non-parametric hashing. We have also presented a unique method of speaker clustering using canopy clustering. When combined with hashing, these methods proved to be a fast and effective way of clustering data. The trade-off between computational efficiency and clustering performance was explored with adjusted mutual information versus computational distance sparsity plots. Future work will explore applying these techniques to other modalities such as clustering audio and video data.

Since the non-parametric hashing was constructed with defined tolerance intervals or hyperspheres, we conjecture that the non-parametric hashing method should be a non-biased estimator of the underlying density function of the input data. Future work will endeavorer to prove this by extending the proof of [8] to non-parametric hashing.

6. References

[1] W. M. Campbell, D. E. Sturim, and D. A. Reynolds, "Support vector machines using GMM supervectors for speaker verification," *submitted to IEEE Signal Processing Letters*, 2005.

[2] N. Dehak, P. Kenny, R. Dehak, P. Ouellet, and P. Dumouchel, "Front end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech and Language Processing*, 2010.

[3] W. M. Campbell, "Generalized linear discriminant sequence kernels for speaker recognition," in *Proceedings of ICASSP*, 2002, pp. 161–164.

[4] W. M. Campbell and E. Singer, "Query-by-example using speaker content graphs." in *INTERSPEECH*. Citeseer, 2012, pp. 1095–1098.

[5] R. Leary and W. Andrews, "Random projections for large-scale speaker search." in *INTERSPEECH*, 2014, pp. 66–70.

[6] L. Schmidt, M. Sharifi, and I. Lopez Moreno, "Large-scale speaker identification," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1650–1654.

[7] W. Jeon and Y.-M. Cheng, "Efficient speaker search over large populations using kernelized locality-sensitive hashing," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4261–4264.

[8] D. O. Loftsgaarden, C. P. Quesenberry *et al.*, "A nonparametric estimate of a multivariate density function," *The Annals of Mathematical Statistics*, vol. 36, no. 3, pp. 1049–1051, 1965.

[9] E. A. Patrick, *Fundamentals of pattern recognition*. Prentice-Hall Englewood Cliffs, New Jersey:, 1972.

[10] Z. Karam and W. M. Campbell, "Graph embedding for speaker recognition," in *Proc. Interspeech*, 2010, pp. 2742–2745.

[11] D. A. van Leeuwen, "Speaker linking in large data sets," in *Proc. Odyssey*, 2010.

[12] M. Ferras and H. Bourlard, "Speaker diarization and linking of large corpora," in *Proceedings of the IEEE Workshop on Spoken Language Technology*, no. EPFL-CONF-192414, 2012.

[13] S. H. Shum, W. M. Campbell, and D. A. Reynolds, "Large-scale community detection on speaker content graphs," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 7716–7720.

[14] K. Greenfield and W. M. Campbell, "Link prediction methods for generating speaker content graphs," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 7721–7725.

[15] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000, pp. 169–178.

[16] A. Rajaraman, J. D. Ullman, J. D. Ullman, and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press Cambridge, 2012, vol. 1.

[17] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 380–388.

[18] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.

[19] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios, "Nearest neighbor retrieval using distance-based hashing," in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, 2008, pp. 327–336.

[20] A. F. Martin and C. S. Greenberg, "The NIST 2010 speaker recognition evaluation," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[21] J. Borgstrom, W. Campbell, N. Dehak, R. Dehak, D. Garcia-Romero, K. Greenfield, A. McCree, D. Reynolds, F. Richardson, E. Singer, D. Sturim, and P. Torres-Carrasquillo, "MITLL 2012 Speaker Recognition Evaluation System Description," in *NIST Speaker Recognition Evaluation*, Orlando, Dec. 2012.

[22] P. Torres-Carrasquillo, N. Dehak, E. Godoy, D. Reynolds, F. Richardson, S. Shum, E. Singer, and D. Sturim, "The MITLL NIST LRE 2015 Language Recognition System," in *Odyssey: The Speaker and Language Recognition Workshop*, 2016.

[23] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: is a correction for chance necessary?" in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1073–1080.