Query-by-Example using Speaker Content Graphs

W. M. Campbell, E. Singer

MIT Lincoln Laboratory, Lexington, MA
\{wcampbell,es\}@ll.mit.edu

Abstract

We describe methods for constructing and using content graphs for query-by-example speaker recognition tasks within a large speech corpus. This goal is achieved as follows: First, we describe an algorithm for constructing speaker content graphs, where nodes represent speech signals and edges represent speaker similarity. Speech signal similarity can be based on any standard vector-based speaker comparison method, and the content graph can be constructed using an efficient incremental method for streaming data. Second, we apply random walk methods to the content graph to find matching examples to an unlabeled query set of speech signals. The content-graph based method is contrasted to a more traditional approach that uses supervised training and stack detectors. Performance is compared in terms of information retrieval measures and computational complexity. The new content-graph based method is shown to provide a promising low-complexity scalable alternative to standard speaker recognition methods.

Index Terms: speaker recognition, graphs

1. Introduction

Speaker recognition is currently an efficient, effective tool in part because of vector-based recognition methods. Techniques such as polynomial expansions, GMM supervectors [1], and iVectors [2] enable application of matrix-based methods for quick and accurate recognition via inner products.

Fundamentally, converting an utterance to a vector involves finding the critical parameters that characterize the speaker information in speech. Additionally, the resulting mapping should be accompanied by a distance, typically Euclidean, that allows quick comparison of mapped vectors with a distance relevant to speaker recognition. For instance, in [1] an approximate KL divergence between MAP-adapted GMM UBMs leads to an inner product and distance between GMM supervectors. Alternately, the iVector approach uses a cosine distance motivated by a Gaussian distribution assumption [2].

Mapping of a corpus of speech utterances to a set of vectors naturally leads to questions about the structure of the underlying manifold. First attempts at modeling the structure of the manifold involved linear subspaces and have been quite successful. Nuisance attribute projection [1] modeled channel effects and session variation as linear subspaces and removed those dimensions. Subsequent methods such as Joint Factor Analysis [2] modeled both channel and speaker variation as subspaces. Finally, the iVector method [2] performs all analysis in a “total variability” subspace.

More advanced methods of analyzing the manifold generated by mapping a corpus to a vector set can be obtained by considering more sophisticated models. One such technique is to “lay” a graph over the manifold and use the resulting graph for analysis and recognition. In the paper by Karam [3], graph embedding is proposed as a method of visualizing large data sets and also as a starting point for speaker comparison (1-1 comparison). A subsequent paper [4] refines the speaker comparison recognition method using relational learning.

In this work, we significantly expand the scope of speaker recognition using graphs. We consider the general problem of query-by-example (QBE). For this problem, we assume we are given a query set of utterances. The goal is to find speakers in a larger corpus that match the query set. This problem includes the standard multi-utterance enrollment multi-speaker detection task as well as speaker comparison. QBE also includes cases where an information retrieval system provides suggestions based upon a user’s browsing history. In contrast to standard speaker recognition, QBE does not assume labeling of all speakers in the query set.

The proposed approach for QBE consists of two parts. First, we construct weighted graphs from content using GMM supervectors. The approach is outlined in Sections 2 and 3. An incremental method for constructing the graphs is presented. Second, once the content graph is constructed, the QBE results are computed by constructing a random walk process on the content graph. Details and efficiency of the process are presented in Section 4. Experiments on the methods presented are shown on a NIST 2008 speaker recognition evaluation (SRE) multi-utterance enrollment task in Section 5.

2. Inner Product Speaker Comparison

Our starting point is speaker comparison—given two speech signals $x$ and $y$ produce a measure of speaker similarity between the two utterances. This process can be accomplished by first extracting a sequence of features vectors (cepstral coefficients and derived features) from the signals. Next, we MAP adapt the means and ML adapt the mixture weights of a GMM UBM to the two utterances separately to obtain two GMM distributions $g_x$ and $g_y$. The adapted GMM means $\mathbf{m}_x$ and $\mathbf{m}_y$ are (super)vectors which encapsulate the speaker content of the two utterances.

The two distributions $g_x$ and $g_y$ can be compared using KL-divergence, but the result is not computable in closed form. An approximate symmetrized KL divergence is used as an alternative [1, 5],

\[
d^2(\mathbf{m}_x, \mathbf{m}_y) = \sum_{i=1}^{N_m} \lambda_{g_x}^2 \lambda_{g_y}^2 (\mathbf{m}_{x,i} - \mathbf{m}_{y,i})^T \Sigma_i^{-1} (\mathbf{m}_{x,i} - \mathbf{m}_{y,i}),
\]

where $\mathbf{m}_{x,i}$ is the vector of means corresponding to the $i$th mixture component, $\lambda_{g_{x,y}}$ is the adapted mixture weight of the $i$th
mixture component, and \( \Sigma_i \) is the diagonal covariance of the ith mixture of the GMM UBM. A corresponding inner product to this distance is

\[
C(\mathbf{m}_x, \mathbf{m}_y) = \sum_{i=1}^{N_m} \lambda_{x,i} \lambda_{y,i} \mathbf{m}_{x,i}^T \Sigma_i^{-1} \mathbf{m}_{y,i}, \tag{2}
\]

and the corresponding SVM expansion is

\[
b(\mathbf{m}_x) = \left[ \lambda_{x,1} \Sigma_1^{-1} \mathbf{m}_{x,1}^T \cdots \lambda_{x,N_m} \Sigma_{N_m}^{-1} \mathbf{m}_{x,N_m}^T \right]^T.	ag{3}
\]

Our methods in this paper are general and are not specific to GMM supervectors and the approximate KL-divergence metric. Alternatively, a polynomial (GLDS) vector or an iVector [2] can be used.

3. Content Graphs

3.1. Constructing Content Graphs

Our approach to constructing graphs is based on the semi-supervised learning literature [6, 7]. We start with a set of vectors \( M = \{ \mathbf{m}_i \} \) obtained as a vector mapping of a corresponding set of speech signals. We imagine that the vectors occupy some manifold. A fundamental idea is to create a graph \( G \) which reflects the local connectivity and distances of points on the manifold. After creating this graph, we can perform fundamental operations such as finding nearest neighbors of points or computing global properties such as geodesic distances using graph methods (Dijkstra’s shortest path algorithm).

The construction of a speaker content graph proceeds as follows. First, each node \( n \) in the graph corresponds to a single vector \( \mathbf{m} \) from a speech signal. We define weights for edges in the graph via a weight matrix \( W = [W_{i,j}] \) where

\[
W_{i,j} = \begin{cases} 
e^{-d^2(m_i, m_j)/\sigma^2} & \text{if an edge exists between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases} \tag{4}
\]

We note that (4) is a standard radial basis function (also commonly used as an SVM kernel). The parameter \( \sigma \) controls the decay of the exponential function (i.e., the width of the basis function). In our methods, we use an approximate KL divergence (1) for the distance in (4).

In order for the graph to reflect the local neighborhood of a point \( \mathbf{m} \), we connect only a limited number of neighbors. One possible method is to choose a fixed threshold \( \epsilon \) and connect all points within an \( \epsilon \)-ball using the distance function \( d(\mathbf{m}_i, \mathbf{m}_j) < \epsilon \). For our experiments, we found this method to yield graphs which were too dense and had unusual degree distributions (the degree is the number of edges incident on a node). Alternatively, for each point we can find the K-nearest neighbors; i.e., for a given node \( n \) with corresponding vector \( \mathbf{m} \), connect \( n \) and \( n_i \) only for the smallest \( K \) values of \( d(\mathbf{m}_i, \mathbf{m}) \) as \( \mathbf{m} \) ranges over the entire vector set \( M \). Note that this construction implies the minimum degree of each node is \( K \), but because the edge construction is done independently for each node, the degree could be substantially larger than \( K \).

Figure 1 shows an example of a content graph using the approximate KL distance and a top-5 nearest neighbor construction. The graph is created from four speakers and we see the corresponding clusters in the graph. The graph clearly shows that not all nodes within a cluster are connected (i.e., not a complete subgraph). This property shows that finding the nearest neighbors is not enough in a search process.

The graph structure has a corresponding isomorphism to a matrix representation. The weighted adjacency matrix \( W \) specified in (4) captures the node connectivity. The matrix \( W \) is sparse and symmetric by construction. In fact, we can view the process of graph construction in the matrix domain as a sparse approximation to the dense matrix where none of the weights are zeroed out in (4).

3.2. Incremental Construction

Content graphs can be constructed in an incremental manner. That is, if we view speech data as streaming in over time, then the content graph can be updated dynamically to reflect the new data. We briefly describe two steps to make this an efficient process.

As a setup, suppose we have a new piece of data \( \mathbf{m} \) and an existing data set \( M = \{ \mathbf{m}_i \} \), content graph \( G \), list of indices and closest \( K \) distances \( \{D_i\} \), and weight matrix \( W \). Note the closest distances for vector \( \mathbf{m} \) in increasing order are \( D_{i,k}, k = 1, \ldots, K \). The process of adding a new piece of data to the graph is equivalent to appending a row and column to \( W \). The steps to adding to the content graph are the following:

- Compute and store \( d(\mathbf{m}, \mathbf{m}_i) \) for \( i = 1, \ldots, K \). Store the list of indices and sorted distances \( D \) with distances \( D_1 \leq D_2 \leq \cdots \leq D_K \).
- For each \( i > K \), compute \( d(\mathbf{m}, \mathbf{m}_i) \) using an early out algorithm. First, retrieve the furthest neighbor distance \( D_{i,K} \) for \( \mathbf{m} \). Then, loop over the dimensions in the Euclidean norm in (1). The current “estimate” of the distance is monotonically increasing; if this estimate goes over \( D_G \) or \( D_{i,K} \), we can stop and go to the next point. Otherwise, insert the point in the appropriate list.
- At the end, we will have the \( K \) closest points to \( \mathbf{m} \) in the list \( D \). We potentially have also inserted \( \mathbf{m} \) into the distances and lists for each of the \( \mathbf{m}_i \).

Several comments on size and computation are appropriate. First, since we need to store only the top \( K \) distances and indices for each \( \mathbf{m}_i \), the storage for the graph is approximately \( 2KN_e \) scalars where \( N_e \) is the number of vectors. Storage increases linearly with the number of vectors. Second, computation to insert an element into the graph is approximately \( 2cN_eN_f \) flops where \( N_f \) is the dimension of the expansion vector and \( c \) is the early out probability (typically 0.5 in our experiments). Essentially one could view this computation as an extreme form of feature reduction performed when the data is streamed into the system. Additionally, the graph construction performs most of the necessary computation up front to avoid significant computation at query time.
We note that in our experiments it was also critical to sphere (normalize) the vector expanded data (3) as \( b(m_s)/\|b(m_s)\|_2 \). Sphering the data partly corrects for variations in vector length due to different signal durations [8].

4. Query-by-Example

4.1. Basic Ideas and Standard Approach

We assume that we have a set of query speech signals represented as vectors \( Q \) contained in a larger set of vectors \( M \) with top-\( K \) graph \( G \) and corresponding weighted adjacency matrix \( W \). Our goal is to find a list of \( N \) candidate matches from \( M \setminus Q \) (set difference) to the query set \( Q \).

The baseline approach to QBE using standard speaker recognition techniques is a stack detector [9]. First, assume that the query set has \( S \) labeled speakers. Then, for each of the speakers construct an SVM model, \( w_s \). Next, apply the models \( \{w_s\} \) as a stack detector to the vectors \( M \setminus Q \). The stack detector score for vector \( m_i \) is \( s_i = \max_j w_j^T m_i \). Typically, the stack detector score would be compared to a threshold for verification. In the QBE example, we sort the scores \( \{s_i\} \) and find the largest \( N \) values and return the corresponding indices as the result of the query.

4.2. QBE with Random Walks

QBE can be performed with a random walk (Markov process) on the content graph constructed in Section 3. As in the previous section, we start with a set of query vectors, \( Q \). Our goal is to set up a random process on the content graph with walks from the query nodes to good candidates with high probability.

A walk is defined as a sequence of nodes on the content graph where a transition between nodes can occur only if there is an edge connecting the nodes. A random walk is a random process \( \{v_t\} \) where each \( v_t \) is a node, and each sample of the process is a walk. Additionally, the random walk is constrained to have the Markov property \( p(v_{t+1}|v_1, \ldots, v_t) = p(v_{t+1}|v_t) \) and is reversible. For details on the theoretical aspects of random walks, we refer the reader to Lovász [10].

The next setup step for the random walk is to define the transition probabilities. We define the diagonal matrix \( D_W \) to have entries

\[
D_{W,i,i} = 1 + \sum_j W_{i,j}. \tag{5}
\]

We then define a transition matrix \( T = (W + I)D_W^{-1} \) where \( W \) is the weight matrix in (4); i.e., the Markov transition probabilities are \( p(v_{t+1} = j|v_t = i) = T_{i,j} \). The intuition behind this construction is that vectors with similar content will have a high transition probability between them. Note that we include a node self-loop transition by adding the identity to the weight matrix.

QBE is accomplished by computing the probability of arriving at the \( j \)th node after \( L \) steps from the query nodes. This calculation can be computed iteratively using a matrix multiply. If we define the vector \( P_0 \) with entries \( P_{0,i} = p(v_0 = i) \), then by the Markov property \( P_{t+1} = TP_t \). Thus, \( P_L = T^LP_0 \) where \( P_0 \) is the starting (prior) distribution. For the query process, we set \( P_{0,i} = 1/|Q| \) if \( q_i \) is in the query set and zero otherwise (\( |Q| \) is the number of elements in \( Q \)).

In addition to the random walk on the graph, we include a restart that returns to the query nodes. This is equivalent to adding edges from every node back to the query node set. With this modification, our update equation becomes

\[
P_{t+1} = \alpha P_0 + (1 - \alpha)TP_t, \tag{6}
\]

where \( 0 \leq \alpha \leq 1 \). For \( \alpha \) greater than 0, the restart term reinforces the query.

4.3. Discussion

A first observation is that the stack-detector method and random walk on a content graph have a fundamentally different philosophy. For the stack-detector method, the approach uses supervised learning and applies standard detection theory to return the query result. The stack-detector method with an SVM assumes knowledge of both labeled-target and non-target data to construct the models. For the random walk method, the fundamental setup is corpus based. Only selection of the positive (query) examples is needed for search.

Computation for the stack-detector and random walk is structured differently. For the stack-detector, the computation at query time is \( 2SNcNK \) flops where \( S \) is the stack size, \( N_c \) is the expansion size, and \( N_e \) is the number of vectors in the corpus. For the random walk, the computation is determined by \( L \) sparse matrix multiplications and is \( 2LNcNK \). Thus, the ratio of computation at query time for the stack-detector to random walk is \( \frac{SN}{LN} \). For small \( S \) and \( L \), this formula implies that gain in speed will be dominated by \( N_e/K \). Depending on the expansion size, this could be several orders of magnitude.

The random walk method presented is inspired by approaches commonly used for text processing or presented as general machine learning methods. Random walks are commonly used in semi-supervised learning for inference, see [11, 12]. Additionally, random walk methods can be viewed as related to using homophily, relational autocorrelation, or label propagation techniques in relational learning [13]. In both cases, the goal is to exploit the fact that nodes “close” to the query nodes will have similar labels.

5. Experiments

5.1. Baseline System

Experiments were performed on the NIST 2008 speaker recognition evaluation (SRE) data set. All telephone data was used resulting in 5,580 male utterances and 9,496 female utterances. Enrollment lists for speaker models were taken from the 8 conversation training scenario from the SRE. Standard feature extraction with 20 MFCCs (including \( c_0 \)) plus delta was performed along with SAD and 0/1 feature normalization. A GMM UBM with 512 mixtures was trained using a large set of Switchboard 2 and Fisher data. For MAP adaptation, a relevance factor of 0.01 was selected. WNAP was trained using a combined NIST SRE 2004-2006 list with the dimension of the nuisance subspace fixed at 64.

SVM speaker models were trained using the kernel in (2) and the methods in [5]. A subset of the Fisher corpus (approximately 4000 utterances) was used as an SVM background. The SVM models were applied using a stack detector with stack sizes \( S = 10, 50 \), and 100. For each stack size, 500 combinations of models were selected at random and scored against all of the NIST SRE test data.

5.2. QBE System Tuning on the Male Speakers

We implemented the random walk system using the methods described in Section 4 and applied it to the NIST data set using the same enroll/test protocol as the stack models for the baseline system. Content graphs per gender were constructed with all of the NIST data. Then, the probabilities of random walks from the utterances of a query set to other nodes on the graph were computed with equation (6).

Performance for the results is compared using average pre-
Table 1: Number of steps \( L \) with best average precision at \( N \) with the random walk method on the male NIST SRE 2008 telephone database.

<table>
<thead>
<tr>
<th>Stack Size ( S )</th>
<th>Query Result Length ( N )</th>
<th>Best ( L )</th>
<th>AvgP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>1</td>
<td>99.24</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>5</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>3</td>
<td>94.76</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>3</td>
<td>72.69</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td>5</td>
<td>99.75</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td>3</td>
<td>99.75</td>
</tr>
</tbody>
</table>

Table 2: Results on female speaker set; comparison with optimal \( L \) and average precision in \( \% \).

<table>
<thead>
<tr>
<th>Stack Size ( S )</th>
<th>Query Result Length ( N )</th>
<th>Best ( L )</th>
<th>Tune AvgP</th>
<th>Best AvgP</th>
<th>Stack AvgP</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>4</td>
<td>98.68</td>
<td>99.04</td>
<td>94.36</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>2</td>
<td>100</td>
<td>100</td>
<td>93.92</td>
</tr>
<tr>
<td>100</td>
<td>5</td>
<td>1</td>
<td>100</td>
<td>100</td>
<td>91.84</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>3</td>
<td>95.28</td>
<td>95.34</td>
<td>90.62</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>5</td>
<td>99.94</td>
<td>100</td>
<td>95.26</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>2</td>
<td>100</td>
<td>100</td>
<td>93.78</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>2</td>
<td>71.69</td>
<td>72.77</td>
<td>71.93</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td>5</td>
<td>99.13</td>
<td>99.91</td>
<td>95.85</td>
</tr>
<tr>
<td>100</td>
<td>20</td>
<td>3</td>
<td>99.99</td>
<td>100</td>
<td>95.13</td>
</tr>
</tbody>
</table>

Precision at a given result list length \( N \); i.e., for a given query, find the top \( N \) scoring nodes and then find the percentage of true trials on the list (the precision). The mean of the precision across multiple queries (stack models) is the average precision.

To facilitate understanding system parameters, we swept the values of \( \alpha = 0.1, 0.1, 10 \) in (4), \( \sigma = 0.1, 0.1, 0.25, 0.5, 0.75 \) in (6), \( K = 2, 5, 10, 25, 50, 100 \) for the top-\( K \) distances described in Section 3, and \( L = 1, \ldots, 5 \) as the number of steps taken in the random walk in (6). We tune to the best average precision on the male subset and apply our system to the female subset in the next section.

We initially considered the number of steps \( L \) in the random walk. Table 1 shows the optimal number of steps to achieve the best average precision. We see that the trend is that for a longer query result list length (\( N \)), more steps are needed to obtain the best precision.

Next we considered the top-\( K \) value for content graph construction. For the cases in Table 1, the optimal value was \( K = 5 \) for 6 out of the 9 cases. A possible reason for the system preferring a smaller \( K \) is that it reduces the number of false alarm connections. The random walk process can recover from this by using more than one step to find a good candidate.

We also tested values of \( \alpha \) and \( \sigma \) for male speakers. For \( \sigma \), the parameter achieving the best average precision for multiple \( S \) and \( K \) was \( \sigma = 1 \); this value was optimal for 6 out of the 9 cases. For \( \alpha \), values of \( 0 \) (4 times), \( 0.1 \) (4 times), \( 0.25 \) (1 time) were found to produce optimal precision. This indicates that very little restarting is needed in the process.

5.3. Application to the Female Speakers

From the previous analysis, we explored two possibilities, a fixed parameter setting and the optimal setting. For the fixed parameter setting, we used \( \alpha = 0.1, \sigma = 1, K = 5 \) based on consideration of the optimal choices for males; for \( L \), we chose a compromise value of 3 based on Table 1. Results for the male tuned system applied to the female data, the optimal system, and the stack system with SVM models are shown in Table 2.

Table 2 shows that the random walk is relatively robust to parameter settings. The tuning parameters were applied in the algorithm to an unseen data set. Additionally, the tuned and optimal average precision are close. Finally, the random walk is performing well in comparison to a standard stack approach. This fact is significant since the random walk approach uses much less computation at query time.

6. Conclusions

A new method of performing query-by-example speaker recognition in a large corpus using content graphs and random walks was presented. Experiments showed this method had good precision for retrieval as compared to a standard classification method. Analysis of the new method showed that it could be implemented in an efficient manner on a large data set. Further research will focus on refining the methods—graph construction, alternate distances, and alternate search methods.

Acknowledgements. Thanks to Glen Coppersmith (JHU HLTCOE) and Carey Piebee (JHU) for introducing us to this problem and providing illuminating discussions on related techniques.

7. References


