Available online at www.sciencedirect.com

ScienceDirect

Speech Communication 66 (2015) 36–46

SPEECH COMMUNICATION

www.elsevier.com/locate/specom

Reducing computation in an i-vector speaker recognition system using a tree-structured universal background model

Richard McClanahan [a,b,*], Phillip L. De Leon [b]

[a] Sandia National Laboratories, Albuquerque, NM, USA
[b] New Mexico State University, Las Cruces, NM, USA

Received 14 March 2014; received in revised form 24 July 2014; accepted 25 July 2014
Available online 20 August 2014

Abstract

The majority of state-of-the-art speaker recognition systems (SR) utilize speaker models that are derived from an adapted universal background model (UBM) in the form of a Gaussian mixture model (GMM). This is true for GMM supervector systems, joint factor analysis systems, and most recently i-vector systems. In all of these systems, the posterior probabilities and sufficient statistics calculations represent a computational bottleneck in both enrollment and testing. We propose a multi-layered hash system, employing a tree-structured GMM–UBM which uses Runnalls' Gaussian mixture reduction technique, in order to reduce the number of these calculations. With this tree-structured hash, we can trade-off reduction in computation with a corresponding degradation of equal error rate (EER). As an example, we reduce this computation by a factor of 15× while incurring less than 10% relative degradation of EER (or 0.3% absolute EER) when evaluated with NIST 2010 speaker recognition evaluation (SRE) telephone data.

Keywords: Speaker recognition; Clustering methods; Tree graphs

1. Introduction

As speaker recognition (SR) systems have become more accurate, there is now a growing volume of research aimed at reducing computational complexity and increasing efficiency (Chan et al., 2004; Dehak et al., 2009; Kinnunen et al., 2004, 2006; McLaughlin et al., 1999; Auckenthaler and Mason, 2001; Xiang and Berger, 2003; Ye et al., 2009; Ye and Mak, 2010; McClanahan and De Leon, 2012; Mohammadi and Saeidi, 2006; Saeidi et al., 2010; Tydlitát et al., 2007; Glembek et al., 2011; Aronowitz and Barkan, 2012). In general, most of this research can be categorized by computational reduction in either the frame, Gaussian, or utterance layer (Chan et al., 2004). Within the frame layer, the focus is usually on reducing the effective rate (i.e. frames per second) at which feature vectors are extracted or passed on for further in the processing chain. Within the Gaussian layer, the focus is usually on reducing the complexity involved in computing the posterior probability or probabilistic *alignment* of each feature vector given the speaker-dependent Gaussian mixture model (GMM) and/or the GMM universal background model (GMM–UBM) which typically contains thousands of component densities. Within the utterance or speaker layer, the focus is usually on fast utterance scoring (Dehak et al., 2009) or efficiently compensating for channel/scoring effects (Kinnunen et al., 2004).

It is worth noting that the techniques applied within the various layers are not mutually exclusive–that is to say that a system might employ frame layer techniques, Gaussian layer techniques as well as utterance layer techniques. In

* Corresponding author at: Sandia National Laboratories, Albuquerque, NM, USA.
E-mail addresses: rmcclan@sandia.gov (R. McClanahan), pdeleon@nmsu.edu (P.L. De Leon).

http://dx.doi.org/10.1016/j.specom.2014.07.003
0167-6393/

this work we focus on Gaussian layer techniques as the calculation of sufficient statistics is an operation that is performed within all recent state-of-the-art systems employing a GMM–UBM including traditional log-likelihood scoring systems (Reynolds et al., 2000), support vector machines (SVM) systems (Campbell et al., 2006), joint factor analysis systems (JFA) (Kenny et al., 2007), and more recently i-vector systems (Dehak et al., 2009). Further, this calculation of the probabilistic alignment is a recognized bottleneck and as such worthy of continued research.

1.1. Frame layer techniques

It has long been recognized that with GMM–UBM based SR systems, a reduction in frame rate reduces the number of feature vectors that must be aligned for the given GMM. Typically frame rate reduction is performed at the test stage as opposed to the training stage, where training times may not be as important as test times. McLaughlin et al. (1999) analyzed the effects of three different decimation approaches: fixed-rate, variable-rate, and adaptive-rate decimation. In fixed-rate decimation, every Dth feature vector is retained from the sequence of vectors. In variable-rate decimation, a difference threshold between successive feature vectors is computed and only vectors which are sufficiently different are retained. In adaptive-rate decimation, only a fixed number of feature vectors per test utterance are retained. Fixed-rate decimation by up to a factor of 20× degraded equal-error rate (EER) by less than 2% (absolute) on the NIST 1998 summer development evaluation data using a GMM–UBM system.

In Kinnunen et al. (2006) the authors analyzed four different pre-quantization (PQ) algorithms: decimation (as in McLaughlin et al. (1999)), random subsampling, averaging, and clustering-based PQ. In random subsampling, 1-out-of-D feature vectors is randomly selected and retained while averaging uses the centroid of D feature vectors. In clustering-based PQ, feature vectors are clustered using the Linde–Buzo–Gray (LBG) algorithm. The centroids of the clusters are then scored in an initial speaker pruning stage (which is a speaker layer technique). In clustering-based PQ, the clustering algorithm itself incurs substantial computational overhead and for many SR implementations, this overhead negates any computational reduction. With a 256 component GMM–UBM system, the clustering PQ algorithm produced a 34× speed-up in speaker identification with a 1.6% absolute degradation in EER on the NIST 1999 speaker recognition evaluation (SRE) corpus.

1.2. Gaussian layer techniques

Some of the earliest research focusing on reducing computation in SR systems was directed at the Gaussian layer and in particular, investigating the impact of using GMMs with fewer component densities. McLaughlin et al. (1999) studied the effects of varying the number of component densities and was able to achieve a 4× speed up with less than 1% absolute degradation in EER by reducing the number of components in the GMM–UBM to 512 from 2048.

Auckenthaler and Mason (2001) used Gaussian selection to reduce the number of component densities that each feature vector was aligned against. This work analyzed three different hash systems. These hash systems employed a hash GMM–i.e. a GMM with much fewer components than the UBM, followed by a limited search of the UBM. Each component of the hash GMM had a shortlist of indices to be searched in the otherwise conventional UBM. Two of the systems trained a smaller hash GMM using the same training data as used to train the 1024 component GMM–UBM. In one system, the shortlist was trained by aligning feature vectors to both the GMM–UBM and the hash GMM and calculating a histogram of highest scoring pairs. In the second system, the method for training is similar to the first except that each UBM component was limited to being linked to one component within the hash. In the third hash system, the authors created a hash model by performing vector-quantization (VQ) on the GMM–UBM mean vectors. With the hash, this technique yielded a 16× computational reduction with "no noticeable" performance degradation as compared to their baseline GMM–UBM system tested with the Odyssey 2001 evaluation data.

Similarly Xiang and Berger (2003) utilized a hierarchical GMM (HGMM) with up to four layers where each layer represents a GMM with an increasing number of components or increasing *resolution*. A tree-structure was formed by applying k-means clustering within the various layers of the tree. In scoring, feature vectors are aligned against the lower resolution GMM, with each component density linked to multiple component densities in the next layer consisting of a higher-resolution GMM. A structural background model (SBM), a HGMM, is searched by iteratively following the highest-scoring components and their children in each layer. The structural GMM for the target speaker is scored using only the path that is taken in the search down the SBM. Finally, the SR decision is made using a neural network that processes scores from different layers. This technique yielded a 17× computational reduction with a 5% relative degradation in EER evaluated with the NIST 1999 SRE.

A different approach, taken by Ye and Mak employs scalar-quantization (SQ) and discrete densities for feature vector alignment to the GMM (Ye et al., 2009). The elements of the feature vector are each scalar-quantized to a codeword which is then used to index a pre-computed discretized probability. The authors investigated using both a high-density discrete model (HDDM) and a discrete mixture model (DMM). The HDDM method applied to a GMM–UBM system yielded a 30× reduction in computation with a 1.5% absolute degradation in EER evaluated using NIST 2002 SRE data. In Ye and Mak (2010) the work was extended to use subvector quantization of the

feature vectors and achieved up to 25× speedup with "no significant" degradation in EER.

In our earliest work McClanahan and De Leon (2012) we presented a hash-based system similar to Auckenthaler and Mason (2001) but using Runnalls' Gaussian mixture reduction (Runnalls, 2007) approach to construct the hash and shortlist. We demonstrated a 2.8× reduction in alignment calculations using a GMM supervector, support vector machine (SVM) system (Campbell et al., 2006) without *any* loss in accuracy on the NIST 2002 SRE.

1.3. Efficient i-vector extraction techniques

SR systems based on i-vectors have recently been developed and investigated (Dehak et al., 2009) and there is now a growing number of papers describing methods for reducing computation involved in i-vector extraction. In Glembek et al. (2011) proposed two methods for simplifying i-vector extraction. The first simplification assumes GMM component alignment, i.e. zeroth-order sufficient statistics, are constant across utterances and that this alignment is represented by the GMM–UBM component weights. The second simplification assumes that the i-vector extraction matrix is linearly transformed such that its per-Gaussian components are orthogonal. This assumption of orthogonality allows a simplified inverse on the diagonal precision matrix. Aronowitz and Barkan (2012) extended this work by implementing two other i-vector extraction approximations assuming GMM counts to be similar across sessions. This technique yielded a 25× speed up with minimal degradation in EER evaluated with NIST 2008 SRE data.

Cumani and Laface (2013, 2014) focused on reducing the memory footprint of fast i-vector extractors. The authors presented several algorithms including a variational Bayes approach, a conjugate gradient approach, and a sub-space factorization method. Most recently, with their sub-space approach, the authors achieved dramatic memory reduction while maintaining fast extraction and little degradation when evaluated with the NIST 2010 extended core evaluation.

1.4. Paper outline

In this paper, we extend the use of Gaussian mixture reduction reported in McClanahan and De Leon (2012) and propose a tree-structured, hash GMM in order to reduce the computation required for alignment and subsequent calculation of the sufficient statistics. This novel method of forming the tree-structured hash GMM has the important practical advantage in that it can be incorporated into a system that has already been trained as it only needs the GMM–UBM itself for training. Although this approach is applicable to adapted GMM–UBM SR systems in general, we focus our attention to reducing computation in state-of-the-art i-vector SR systems. We believe this to be one of the first papers to report results from such a hashing system implemented within an i-vector SR system. We show that using a tree-structured, hash GMM, we are able to achieve a large reduction in computation while incurring a small, relative degradation in EER.

The remainder of the paper is organized as follows. In Section 2, we review the development and implementation of a typical i-vector SR system. In Section 3, we describe how to utilize Gaussian mixture reduction to train the hash GMM and tree-structured, hash GMM. In Section 4, we describe our i-vector system with the tree-structured, hash GMM and present results in Section 5. Finally, in Section 7, we conclude the article.

2. Description of i-vector speaker recognition system

This section reviews the i-vector SR system as described by Dehak et al. (2009, 2010, 2011), Garcia-Romero and Espy-Wilson, 2011 and utilized in this research. Fig. 1 shows a block diagram of the system where the dashed blocks show the steps involved in system development including UBM training, total variability (TV) subspace training, and probabilistic linear discriminant analysis (PLDA). The lower processing chain shows the steps involved in extracting i-vectors for target and test utterances and scoring the similarity between the utterances using the various parameters that were estimated during system development.

2.1. System development

2.1.1. UBM training and sufficient statistics

Development begins with construction of a GMM–UBM of non-target speakers' feature vectors. The GMM is given by

$$p(\mathbf{x}|\lambda) = \sum_{c=1}^{M} w_c p_c(\mathbf{x}) \tag{1}$$

where M is the number of components densities and

$$p_c(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}_c|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_c)^{\mathsf{T}}\mathbf{\Sigma}_c^{-1}(\mathbf{x}-\boldsymbol{\mu}_c)\right) \tag{2}$$

where d is the dimensionality of feature vector $\mathbf{x}$. The parameters of the GMM–UBM are collectively denoted by $\lambda_{\text{UBM}} = \{w_c, \boldsymbol{\mu}_c, \mathbf{\Sigma}_c\}$ indicating the weight, mean vector, and diagonal covariance matrix respectively.

GMM parameters are estimated using the iterative expectation–maximization (EM) algorithm (Reynolds, 2009). The E-step calculates the probabilistic alignment for each feature vector

$$Pr(c|\mathbf{x}_t, \lambda) = \frac{w_c p_c(\mathbf{x}_t|\boldsymbol{\mu}_c, \mathbf{\Sigma}_c)}{\sum_{k=1}^{M} w_k p_k(\mathbf{x}_t|\boldsymbol{\mu}_k, \mathbf{\Sigma}_k)} \tag{3}$$

and the M-step re-estimates the sufficient statistics

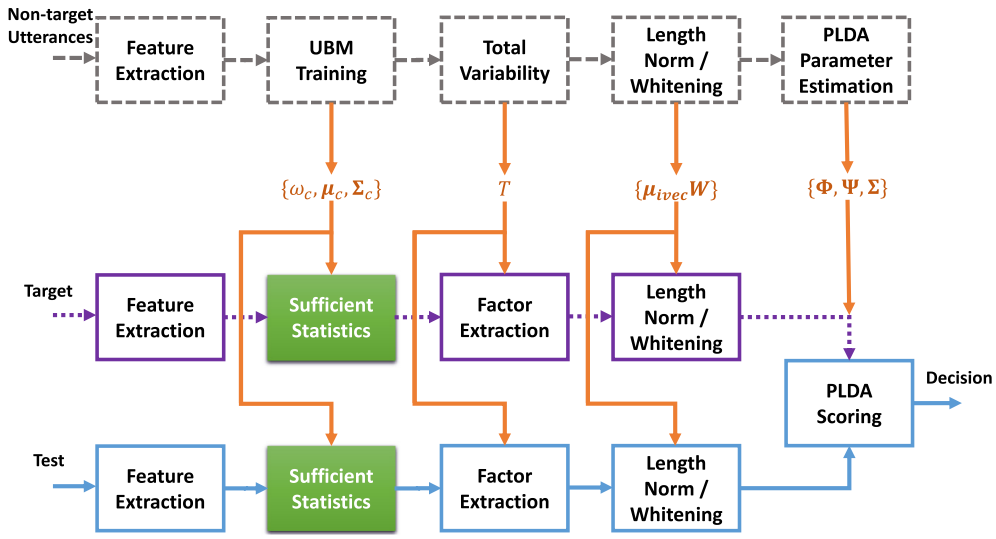

Fig. 1. Baseline i-vector system including development phase. The dashed blocks in the top flow represent those steps necessary for system development and parameter training. The lower blocks represent the target model and test model extraction and scoring. The sufficient statistics blocks are shaded to highlight the portion of the processing chain that we are focusing on for computational reduction using our hashing system.

$$\hat{w}_c = \frac{1}{n_t}\sum_{t=1}^{n_t} Pr(c|\mathbf{x}_t, \lambda), \tag{4}$$

$$\hat{\boldsymbol{\mu}}_c = \frac{\sum_{t=1}^{n_t} Pr(c|\mathbf{x}_t, \lambda)\mathbf{x}_t}{\sum_{t=1}^{n_t} Pr(c|\mathbf{x}_t, \lambda)}, \tag{5}$$

and

$$\hat{\boldsymbol{\Sigma}}_c = \frac{\sum_{t=1}^{n_t} Pr(c|\mathbf{x}_t, \lambda)\mathbf{x}_t^2}{\sum_{t=1}^{n_t} Pr(c|\mathbf{x}_t, \lambda)} - \hat{\boldsymbol{\mu}}_c^2 \tag{6}$$

where n_t is the number of training feature vectors. The algorithm is iterated until a convergence criterion is met, i.e. a maximum number of iterations is reached or log-likelihood improvement is below a pre-defined threshold.

Though not explicitly shown in the development stage, it is necessary to calculate the sufficient statistics for each training utterance that is used in TV and PLDA. While PLDA operates upon i-vectors, these i-vectors must first be extracted from sufficient statistics and TV parameters. The zeroth-order and centralized first-order sufficient statistics for each utterance, u are given by

$$N_c(u) = \sum_{t=1}^{n_t} Pr(c|\mathbf{x}_t, \lambda) \tag{7}$$

and

$$\widetilde{\mathbf{F}}_c(u) = \sum_{t=1}^{n_t} Pr(c|\mathbf{x}_t, \lambda)(\mathbf{x}_t - \boldsymbol{\mu}_c). \tag{8}$$

2.1.2. Total variability training

The second step of development is to use the EM algorithm (Kenny et al., 2005) to train the TV matrix, $\mathbf{T}$ (rank R and size $dM \times R$) using the sufficient statistics extracted from a large number of training utterances; $\mathbf{T}$ models the speaker, session, and channel variabilities.

In the E-step, we estimate the posterior (normal) distribution, $\mathbf{w}(u)$ based on the current estimates of $\boldsymbol{\mu}, \mathbf{T}$, and diagonal covariance matrix $\boldsymbol{\Sigma}$. The UBM covariance is used as an initial estimate of $\boldsymbol{\Sigma}$. Defining the $R \times R$ precision matrix $\mathbf{P}(u)$ as

$$\mathbf{P}(u) \equiv \mathbf{I} + \mathbf{T}^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}\mathbf{N}(u)\mathbf{T} \tag{9}$$

where $\mathbf{N}(u)$ represents the $Md \times Md$ diagonal matrix with $N_1(u)\mathbf{I}_{d\times d}, \cdots, N_M(u)\mathbf{I}_{d\times d}$ along the diagonal, the mean and covariance of $\mathbf{w}(u)$ are given by

$$E[\mathbf{w}(u)] = \mathbf{P}^{-1}(u)\mathbf{T}^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}\widetilde{\mathbf{F}}(u) \tag{10}$$

where $\widetilde{\mathbf{F}}(u)$ is the supervector of stacked first order statistics $\widetilde{\mathbf{F}}_c(u)$ for utterance u and

$$\text{cov}\,(\mathbf{w}(u), \mathbf{w}(u)) = \mathbf{P}^{-1}(u). \tag{11}$$

In the M-step, we compute

$$\mathbf{C} = \sum_u \widetilde{\mathbf{F}}(u)E[\mathbf{w}(u)] \tag{12}$$

and

$$\mathbf{A}_c = \sum_u N_c(u)E[\mathbf{w}(u)\mathbf{w}^{\mathsf{T}}(u)] \tag{13}$$

where $E[\mathbf{w}(u)\mathbf{w}^{\mathsf{T}}(u)] = \mathbf{P}^{-1}(u) + E[\mathbf{w}(u)]E[\mathbf{w}(u)]^{\mathsf{T}}$ and then update block submatrices of $\mathbf{T}$ with

$$\mathbf{T}(\mathbf{i}, :) = \mathbf{C}_{\mathbf{i}}\mathbf{A}_c^{-1} \tag{14}$$

where $\mathbf{i}$ is a vector of indices ranging from $(M-1)d + 1$ to Md.

2.1.3. PLDA system development

Garcia-Romero and Espy-Wilson (2011) demonstrated a simplified PLDA model that has exceptional performance. The author made two changes to the original PLDA algorithm. First, he introduced whitening and

length normalization to the raw i-vectors. The second change was to eliminate the within-class variability matrix and instead assume a full (not diagonal) covariance matrix.

In this simplified model, the observed length normalized i-vector $\mathbf{w}_i$ for the ith speaker is modeled by

$$\mathbf{w}_i = \mathbf{\Phi}\boldsymbol{h}_i + \boldsymbol{\epsilon}_i \tag{15}$$

where $\mathbf{\Phi}$ is the basis for the between-class variability, $\boldsymbol{h}_i$ are the speaker factors for the ith speaker, and $\boldsymbol{\epsilon}_i$ is vector of residuals generated from a full covariance matrix $\mathbf{\Sigma}$.

2.2. i-Vector extraction

After system development, i-vectors are extracted for testing according to the bottom row of Fig. 1 using parameters estimated from system development. The uncompensated i-vectors are computed using (10), repeated here as

$$\mathbf{w} = \mathbf{P}^{-1}(u)\mathbf{T}^{\mathsf{T}}\mathbf{\Sigma}^{-1}\tilde{\mathbf{F}}(u). \tag{16}$$

2.3. Decision

Following Garcia-Romero and Espy-Wilson (2011) the log-likelihood ratio, or speaker comparison score, computed between two i-vectors–$\mathbf{w}_1$ and $\mathbf{w}_2$–is

$$\text{score} = \mathbf{w}_1^{\mathsf{T}}\boldsymbol{Q}\mathbf{w}_1 + \mathbf{w}_2^{\mathsf{T}}\boldsymbol{Q}\mathbf{w}_2 + 2\mathbf{w}_1^{\mathsf{T}}\boldsymbol{P}\mathbf{w}_2 + \mathcal{C} \tag{17}$$

where

$$\boldsymbol{Q} = \Sigma_{\text{tot}}^{-1} - \left(\Sigma_{\text{tot}} - \Sigma_{\text{ac}}\Sigma_{\text{tot}}^{-1}\Sigma_{\text{ac}}\right)^{-1} \tag{18}$$

$$\boldsymbol{P} = \Sigma_{\text{tot}}^{-1}\Sigma_{\text{ac}}\left(\Sigma_{\text{tot}} - \Sigma_{\text{ac}}\Sigma_{\text{tot}}^{-1}\Sigma_{\text{ac}}\right)^{-1} \tag{19}$$

and where $\Sigma_{\text{tot}} = \mathbf{\Phi}\mathbf{\Phi}^{\mathsf{T}} + \mathbf{\Sigma}$ is the i-vector total covariance matrix, $\Sigma_{\text{ac}} = \mathbf{\Phi}\mathbf{\Phi}^{\mathsf{T}}$, and $\mathcal{C}$ is a constant. Eq. (17) assumes that the i-vectors have been previously length normalized and whitened.

3. Forming tree-structured GMMs via gaussian mixture reduction

3.1. Hash system description

The purpose of our hash system is to reduce the total number of component densities within the UBM that each input feature vector is aligned with as in (3) and reduce the number of components used in the update of sufficient statistics as in (7) and (8). Given a feature vector $\mathbf{x}_t$, for the vast majority of components the posterior probability $Pr(c|\mathbf{x}_t, \lambda)$ will be negligible. The hash system is used to exploit this result by intelligently selecting a subset of components that will produce a non-negligible posterior probability. Thus, (3) will only be calculated for a subset of components determined by the hash. Further, because the hash exploits the result that $Pr(c|\mathbf{x}_t, \lambda)$ will be negligible for the majority of components, we can also skip these terms in the summations in (7) and (8).

In our system, we employ a hash GMM that consists of M_h component densities with each of these components being linked with one or more components in the M component GMM–UBM. Generally, in order to substantially reduce the number of computations involved in the alignment, we require $M_h \ll M$ where M_h is a free parameter selected by the developer.

Fig. 2 illustrates the principle of using a hash system. Individual input feature vectors, $\mathbf{x}_t$, are initially scored against the component densities of the hash GMM with model parameters, $\lambda_{\text{hash}} = \{w_c^{\text{h}}, \boldsymbol{\mu}_c^{\text{h}}, \mathbf{\Sigma}_c^{\text{h}}\}$, representing component weight, mean vector, and covariance matrix respectively, where $1 \leqslant c \leqslant M_h$ and $^{\text{h}}$ denotes hash parameters. Based on this alignment, a subset of component densities within the GMM–UBM having the model parameters, $\lambda_{\text{UBM}} = \{w_c, \boldsymbol{\mu}_c, \mathbf{\Sigma}_c\}$ are selected for scoring and adaptation. The mapping that links components within the hash GMM to components in the UBM is referred to as a *shortlist mapping*.

It should be noted that the hash GMM can be regarded as a lower resolution version of the UBM (Xiang and Berger, 2003). While the UBM statistically models the distribution of the feature vectors, the hash GMM attempts to model this same statistical distribution but using fewer components, hence a lower resolution.

3.2. Hash GMM construction with gaussian mixture reduction

In McClanahan and De Leon (2012) we found that in a GMM supervector support vector machine (SVM) system, a hash GMM–UBM using Runnalls' Gaussian mixture reduction provided least degradation for a given amount of computational reduction when compared to other methods used to build the hash such as k-means clustering or retraining a lower resolution GMM using the developmental

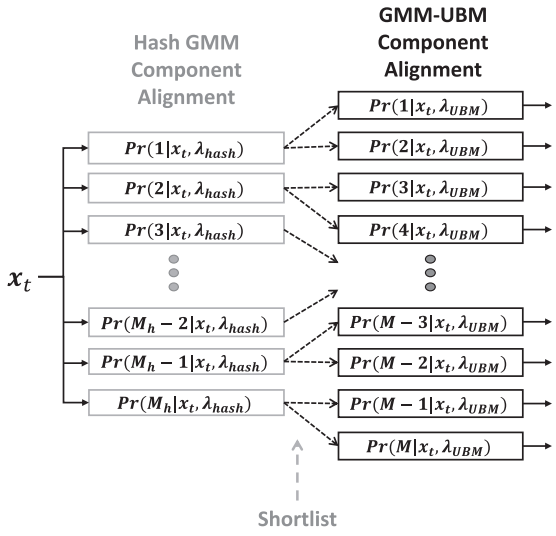

Fig. 2. Single layer hash GMM system trained using Gaussian mixture reduction (McClanahan and De Leon, 2012). Input feature vectors, $\mathbf{x}_t$, are first aligned to the hash GMM components. A shortlist is then used to select a subset of UBM components for alignment based on the best aligned components within the hash.

data used to train the UBM. One benefit of using Gaussian mixture reduction is that the reduction can be applied to a SR system with a pre-existing GMM–UBM. In Gaussian mixture reduction, we start with an M-component GMM and progressively reduce the number of components until we are left with an M_h-component GMM.

In Runnalls (2007) presented an iterative method for successively *merging* a pair of component densities into a single component density. The criterion for selecting the pair to merge is based on minimizing the symmetric Kullback–Leibler (KL) divergence between the *pre-merged* GMM and the resulting *post-merged* GMM. Although a closed-form solution does not exist for calculating the symmetric KL divergence between two GMMs, (Runnalls, 2007) presents an upper bound on the divergence between pre-merged and post-merged GMMs and it is this upper bound that the component selection minimizes.

In the Gaussian mixture reduction method, the following two steps are iterated $M - M_h$ times:

Step 1: Calculate the upper bound of the symmetric KL divergence between the pre-merged GMM and the post-merged GMM (Runnalls, 2007),

$$U(i,j) \leqslant \frac{1}{2}\left[(w_i + w_j)\log\det\left(\boldsymbol{\Sigma}_{ij}\right) - w_i \log\det\left(\boldsymbol{\Sigma}_i\right) - w_j \log\det\left(\boldsymbol{\Sigma}_j\right)\right] \tag{20}$$

for every pair of components i and j within the pre-merged GMM, where $\boldsymbol{\Sigma}_{ij}$ is calculated as shown below in Step 2.

Step 2: Choose the pair of components with indices i and j that minimize (20) and replace with the moment-preserving component density with parameters

$$\begin{aligned} w_{ij} &= w_i + w_j \\ \boldsymbol{\mu}_{ij} &= w_{i|ij}\boldsymbol{\mu}_i + w_{j|ij}\boldsymbol{\mu}_j \\ \boldsymbol{\Sigma}_{ij} &= w_{i|ij}\boldsymbol{\Sigma}_i + w_{j|ij}\boldsymbol{\Sigma}_j + w_{i|ij}w_{j|ij}\left(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\right)\left(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\right)^{\mathsf{T}} \end{aligned} \tag{21}$$

where $w_{i|ij} = w_i/(w_i + w_j)$ and $w_{j|ij} = w_j/(w_i + w_j)$.

Fig. 3 depicts this iterative Gaussian mixture reduction process for a univariate GMM. Starting with a univariate GMM with five components, pairs of components are iteratively merged resulting in a GMM with one fewer component after each iteration. At each iteration of the merging process, a record is kept of which components are merged and later used as the shortlist between the hash GMM and the components of the GMM–UBM.

As was noted in Runnalls (2007) the method chooses a pair of component densities for merging based on the following characteristics:

1. Small weights, w_i and w_j.
2. Mean vectors close to each other with respect to their variances captured by the calculation of $\boldsymbol{\Sigma}_{ij}$.
3. Similar covariance matrices, $\boldsymbol{\Sigma}_i \approx \boldsymbol{\Sigma}_j$.

In the SR test stage, each input feature vector is aligned with the M_h components of the hash GMM. After identifying the B best-aligned components within the hash, the shortlist is used to select clusters or groups of components within the GMM–UBM and the input feature vector is then aligned with those GMM–UBM components. If $B = M_h$ then all the components within the UBM will be used for scoring. If $B \ll M_h$, only a small subset of components within the UBM will be identified for alignment resulting in an overall computational reduction.

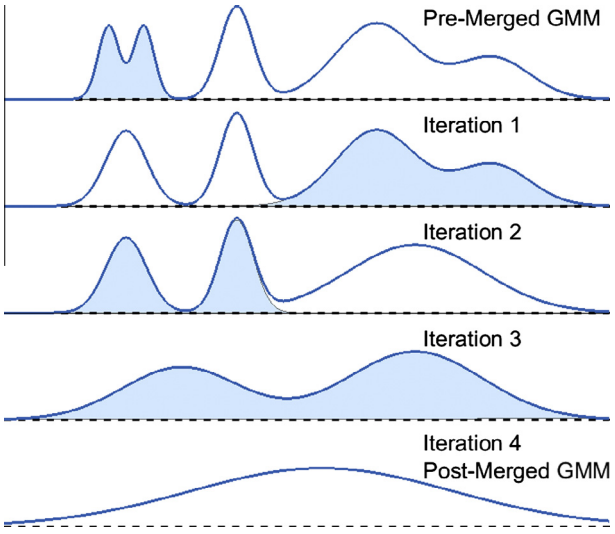

Fig. 3. Demonstration of Gaussian mixture reduction with a five component, univariate GMM. At each iteration, two components (shaded) are merged until only one component remains.

With a single-layer hash system, the upper bound of processing reduction is a factor of $\sqrt{M}/2$ assuming uniformly-distributed cluster sizes. This upper bound is determined as follows. Assuming a uniformly-distributed mapping of GMM–UBM components to hash GMM components, we would perform M_h alignment calculations within the hash system and $(BM)/M_h$ alignment calculations within the GMM–UBM. If we set "$B = 1$"–the value that minimizes the number of alignment calculations–then the total number of calculations is $M_h + M/M_h$ which is minimized when $M_h = \sqrt{M}$. In this case, the total number of alignment calculations in the hashing system would be $2\sqrt{M}$. Thus the reduction is $M/(2\sqrt{M})$ or simply $\sqrt{M}/2$.

While we could reduce the number of alignment calculations within the UBM by choosing a larger hash model (more components on the left side of Fig. 2), the effect is to increase the number of components in the hash for scoring because the hash itself is large. Likewise, choosing a hash model with very few components would result in a larger number of UBM components that are selected for alignment. As such, within a single layer hash system, the best-case reduction can be achieved when the number of components of the hash GMM is equal to the number of UBM components that each hash component maps to.

3.3. Tree-structured hash GMM

We can extend the proposed hash system by applying Gaussian mixture reduction to a multi-layered tree-structured

system as depicted in Fig. 4 to obtain an even greater reduction in the number of alignment calculations. Each layer of the tree structure consists of hash GMMs representing progressively higher resolution models of the underlying GMM–UBM. As depicted in the figure, the UBM components are generally *not* uniformly-distributed among the hash components and thus the tree structure is not balanced along each branch. This non-uniformity occurs because the Gaussian mixture reduction algorithm does not attempt to enforce uniform clustering of component densities. As a consequence, within the different layers, some hash GMMs may be linked to other hash GMMs, to UBM components, or to a mixture of the two.

When aligning input feature vectors with the tree-structure, each input feature vector is aligned with the lowest resolution hash GMM (shown as Layer 1 in Fig. 4). Within each layer, each hash GMM will be linked with M_h other hash GMMs or UBM components via a shortlist. Of the M_h possible paths, we will choose $B < M_h$ by selecting those that maximize the probabilistic alignment of the feature vector with the components of the particular hash GMM that is being traversed. We continue this process through each layer until all branches have been traversed. In Fig. 4, we depict the case with $M_h = 3, B = 1$, and number of layers, $L = 3$. Note that with $B = 1$, we only follow a single path through each layer of the tree until we reach the final layer at which point the hash may point to multiple UBM components (as depicted with the dashed line in Fig. 4).

The recursive tree forming algorithm, which is performed once in the SR development, is presented in Algorithm 1. The algorithm is recursively called until L layers have been created. Essentially, the algorithm is applying Gaussian mixture reduction at different layers within the tree. First, a single pass of Gaussian mixture reduction is applied to the entire set of M UBM component densities resulting in a M_h component hash GMM. Each of these hash components is mapped to components within the UBM. Gaussian mixture reduction is then applied to each of these clusters of components resulting in M_h more hash GMMs each of which consists of M_h components. This algorithm is applied recursively until the desired number of layers has been achieved or there are no longer enough components within the individual clusters to complete Gaussian reduction.

Algorithm 1: Recursive Tree Training Algorithm

Input: GMM components $\{w_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$ passed in from upper layer (or UBM components for top layer), M_h, and remaining number of layers L_R

Output: multi-layered GMM tree structure

Perform Runnalls' Gaussian mixture reduction as using $\{w_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$ creating a smaller hash GMM with M_h components ;

if $L_R = 0$ *(last layer)* **then**
  return GMM clusters from current layer ;
else
  $L_R = L_R - 1$;
  for $1 \leq$ *cluster (node)* $\leq M_h$ **do**
    if *Number of components associated with current node and layer is less than* M_h **then**
      Continue to next node ;
    else
      Call Recursive Tree Training Algorithm with $\{w_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$, M_h, and L_R where k indicates components associated with current node and layer ;
    end
  end
end

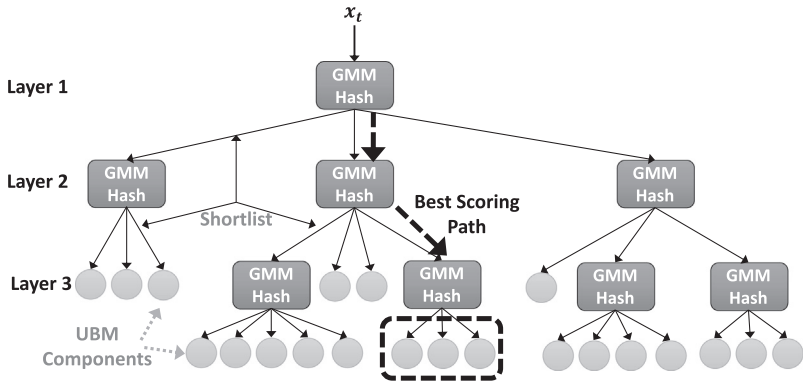

Fig. 4. Tree-structured hash GMM system with each layer trained using Gaussian mixture reduction on the components of the higher resolution GMMs.

We can calculate an upper bound on the computational reduction offered by the tree-structured hash by recognizing that ideally, each hash GMM has two components with each linked to two components of a higher resolution hash, i.e. a binary tree. In this case, the number of layers would be $\log_2(M)$ resulting in $2\log_2(M)$ alignment calculations. As such, the maximum reduction in the computation of alignment calculations would be $M/[2\log_2(M)]$.

As in the case of the single layer system, training the tree-layered hash is an operation that can occur after a typical i-vector system has been developed. Further, because of this coupling to the baseline UBM system, it is possible to insert it into the system or remove it at any time. It would also be possible to change the structure of the hash tree *on-the-fly* which would be useful in situations where the number of verification requests increases quickly and the SR system is required to process rapidly.

4. Experiments

4.1. System description and development

Our reference SR system is based on the i-vector approach as described in Section 2. Referring to Fig. 1, 60-dimensional feature vectors are extracted as follows. We extract 19 mel-frequency cepstral coefficients (MFCCs) using a 25 ms Hamming window with 10 ms advance and append the log-energy. The Δ and $\Delta\Delta$ of these features are then computed. Feature warping is applied to the feature vector using a 3 s window after vectors corresponding to silence have been removed. We construct a single gender-independent GMM–UBM with 1024 components and with diagonal covariance matrices. A single gender-independent i-vector extractor consisting of TV matrix of rank 400 is used. It should be re-iterated that the entire set of parameters for the GMM–UBM, TV matrix, and PLDA were estimated without the use of the hashing system.

Speech signals from several different corpora, were used in system development as shown in Table 1. In our SR system, we did not attempt to prevent overlap in training utterances between different system blocks nor did we attempt to limit our data to a particular channel type such as telephone.

4.2. Evaluation

Experiments were performed using the NIST 2010 SRE data (Martin and Greenberg, 2010). In particular, we focused on condition 5 of the extended core evaluation (tel-tel). Performance was evaluated using the EER. Results for our baseline i-vector SR system are given in Table 2 which shows a pooled 2.90% EER (2.91% EER for male and 2.92% EER for female) without score normalization.

We believe our system without the hash GMM provided adequate baseline results for our research.

Table 1
Corpora used in system development and training.

	UBM	TV	PLDA
Switchboard II, Phase 1 and 3	✓	✓	
Switchboard Cellular, Part 2	✓	✓	✓
Fisher English Part 1 and 2	✓	✓	
NIST 2004 SRE	✓	✓	✓
NIST 2005 SRE	✓	✓	✓
NIST 2006 SRE	✓	✓	✓
NIST 2008 SRE			✓

4.3. Evaluation of i-vector SR system with tree-structured UBM

In our experiments, we measure the effect that a tree-structured UBM has on EER and the resulting computational reduction of sufficient statistics. In the tree-structured UBM, we vary the number of components for each hash GMM M_h, the number of layers L, and the number of best performing paths B that are traversed as described in Section 3.3. A total of 36 different configurations were evaluated but we only present a sampling of the configurations as shown in Table 3.

We evaluate the reduction of sufficient statistics computations by determining the number of alignment calculations performed with the tree-structured UBM versus with the standard UBM. We measure the computational reduction as

$$R = \frac{M}{M_{\text{eff}}} \tag{22}$$

where M_{eff} is the total number of alignment calculations in (7) that are performed in both the calculations within the tree and the subset of original UBM components that are selected from the hash. Using Fig. 4 as an example, following the single best path denoted by the green dashed line, M_{eff} would account for the alignment calculations associated with the three hash GMMs in three layers as well as the three UBM components. This is a conservative metric because while we account for reduction in alignment calculations in our reported reduction, we do not account for the even greater reduction in the calculation of the first-order statistics in (8). For each feature vector $\mathbf{x}_t$, we no longer calculate $\tilde{\mathbf{F}}_c(u)$ for every value of c but rather for only a subset identified by the hash system.

While our primary metric for evaluating our system is that given in (22), we also compute the speed up in CPU time. This speed up is calculated by dividing the total amount of time required to calculate the sufficient statistics

Table 2
Baseline results of the i-vector speaker recognition system for NIST 2010 SRE extended-core condition 5.

	Pooled (%)	Males (%)	Females (%)
EER	2.90	2.91	2.92

Table 3
Results of i-vector system with tree-structured hash GMMs for pooled male/female speakers evaluated with NIST 2010 core-ext condition 5 tel-tel for a sampling of configurations. The bottom row with $(L = 1, M_h = 32, B = 1)$ demonstrates an example single layer configuration. In addition to the computational reduction $R\times$ we also present the actual measured speed up as measured in the reduction in CPU time.

L	M_h	B	EER (%)	R (×)	Speed up (×)	Hash (MB)
Baseline			2.90	1.00	1	0.99
10	2	1	6.33	45.32	8.8	2.00
4	**5**	**1**	**5.11**	**44.47**	**17.3**	**1.39**
3	8	1	4.73	38.69	20.9	1.50
2	**16**	**2**	**3.21**	**15.32**	**16.3**	**1.25**
2	16	1	4.38	27.86	24.5	1.25
2	12	2	3.20	14.54	14.3	1.14
2	12	1	4.55	31.17	24.0	1.14
2	10	1	4.63	32.44	23.6	1.10
2	8	1	4.51	29.33	21.5	1.06
1	32	1	4.07	15.24	21.5	1.02

using the given hashing configuration by the total amount of time required using the traditional GMM system.

Within the i-vector SR system–as shown in Fig. 1–there are four primary blocks: test segment feature extraction, sufficient statistics calculation, i-vector extraction, and PLDA scoring. Within our baseline system, we have found that feature extraction, sufficient statistics calculation, and i-vector extraction each require about a third of the total test stage time while PLDA scoring is negligible in comparison. The results that we report for computational reduction and speed up are strictly for this sufficient statistics calculation not the end-to-end total system.

5. Results

The results for our system vary as a factor of the computational reduction. This reduction factor is determined by logging all of the likelihood calculations performed within the tree structure and comparing this to the baseline of 1024 likelihood calculations. Fig. 5 shows the EER as a function of computational reduction (reduction in alignment calculations) for pooled male and female trials based on 36 different hashing configurations. The general trend is that greater reductions in computational complexity come at the expense of greater EER.

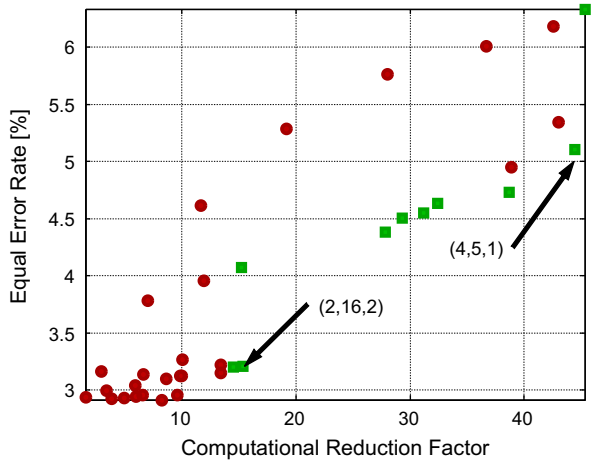

Fig. 5. EER as a function of computational reduction for an i-vector system with tree-structured hash GMMs evaluated with NIST 2010 core-ext condition 5 using pooled male/female results. The green squares depict the results for the configurations listed in Table 3 and other configurations not in Table 3 are depicted by the red circles. The results for the two configurations $(L = 2, M_h = 16, B = 2)$ and $(L = 4, M_h = 5, B = 1)$ are indicated with the arrows. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 3 illustrates results from various configurations of number of layers L, number of components per hash M_h, and number of best scoring paths B that are traversed and the resulting EER and computational reduction. For a computational reduction by a factor greater than $44\times$ $(L = 4, M_h = 5, B = 1)$, absolute EER increases from 2.90% to 5.11%–a 76% relative degradation. Depending on the application, such an increase in EER for the given computational reduction may or may not be acceptable.

For more modest computational reductions, degradations to EER can be reduced. For example as shown in Table 3, using a tree-structure hash UBM with parameters $(L, N, B) = (2, 16, 2)$ we can achieve a factor of $15\times$ computational reduction while incurring a relative degradation in EER of 10%–which corresponds to increasing the absolute EER from 2.9% to 3.21%.

Though not shown in the table, configurations with $B \geqslant 3$ were evaluated. While these configurations incurred the least degradation, none of these configurations were able to achieve at least an order of magnitude reduction in computation.

In analyzing the results in Table 3, the first thing to note is that computation reductions of up to $15\times$ are relatively easy with a number of different configurations with little EER degradation. Above this level of computational reduction, EER degradation increases quickly. Results show that we are able to achieve greater computational reduction with an increased number of layers in the tree up to the theoretical limit of $\log_2(M)$ layers–with the maximum reduction occurring when we have 10 layers (for a 1024 component density GMM–UBM).

From the results, it appears that an acceptable configuration consists of a balanced approach between large hash

GMMs (M_h large) and a large number of layers (L large). With a single layer system ($L = 1$) and with $M_h = 32$, the maximum computational reduction is 15× with 40% relative degradation in EER. At the other extreme with 10 layers and $M_h = 2$, a reduction by 45× is possible with a near doubling in EER. Better results are achieved when the configurations are more balanced such as (L, M_h) equal to $(4, 5)$ or $(3, 8)$ which balance the number of components selected within the UBM with the number of components in the hash system.

It should be noted that these results are not competitive with those presented in Section 1.3 but rather complimentary to them. For example, a system could implement a hashing system for sufficient statistics extraction followed by one of the techniques in Section 1.3.

6. Remarks on memory

While the two metrics we focus on for this research are the EER and computational reduction, it is worthwhile commenting on the memory footprint of the baseline GMM–UBM and the hash GMMs. If all GMM components are represented as double precision floating point numbers, the GMM–UBM–consisting of mean vectors, covariance matrices and component weights–will require $8M(2d + 1)$ bytes of memory–0.99 MB for $M = 1024$ and $d = 60$. Table 3 gives the size of the hash GMM in megabytes (MB) for the various configurations. This number represents the storage requirement for both the hash GMM plus the components of the UBM. Thus, the hash typically requires an additional several kilo-bytes of storage. While this increase in size may seem large, in reality compared to the size of the TV matrix which can require tens or hundreds of MB, storage requirements for the hash are relatively small.

7. Conclusion

Incorporation of a tree-structured hash GMM into an i-vector framework for speaker recognition was proposed and analyzed in this study. Within a vast majority of speaker recognition systems currently in use–including GMM, SVM supervector, joint factor analysis, and i-vector systems–calculation of sufficient statistics and statistical model adaptation require substantial computational resources. We have shown that it is possible to significantly reduce this computation through the use of a tree-structured hash UBM. With this approach, we can achieve a factor of greater than 15× reduction in probabilistic alignment (posterior probability) calculations within the sufficient statistics calculation stage while incurring a *relative* degradation of approximately 10% (or 0.3% absolute). Further gains by as much as a factor of approximately 44 can be achieved but incurring a relative degradation of about 76% (2.21% absolute).

Further, the method presented uses Gaussian mixture reduction to construct a tree-structured hash using only the GMM UBM itself and thus can be constructed post development of the i-vector system. As the hash is compatible with the existing UBM system, it would be possible to switch in and out various hashing configurations *on-the-fly* as computational loads vary within the overall system.

References

Aronowitz, H., Barkan, O., 2012. Efficient approximated i-vector extraction. In: Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), pp. 4789–4792.

Auckenthaler, R., Mason, J.S., 2001. Gaussian selection applied to text-independent speaker verification. In: Proc. IEEE Speaker and Language Recognition Workshop (Odyssey), pp. 83–88.

Campbell, W.M., Sturim, D.E., Reynolds, D.A., 2006. Support vector machines using GMM supervectors for speaker verification. IEEE Signal Proc. Lett. 13 (5), 308–311.

Chan, A., Sherwani, J., Mosur, R., Rudnicky, A., 2004. Four-layer categorization scheme of fast GMM computation techniques in large vocabulary continuous speech recognition systems. In: Proc. Int. Conf. Spoken Language Process. (ICSLP), 2004, pp. 289–292.

Cumani, S., Laface, P., 2013. Memory and computation trade-offs for efficient i-vector extraction. IEEE Trans. Audio Speech Language Process. 21 (5), 934–944.

Cumani, S., Laface, P., 2013. Fast and memory effective i-vector extraction using a factorized subspace. In: Proc. Int. Speech Commun. Association (Interspeech).

Cumani, S., Laface, P., 2014. Factorized sub-space estimation for fast and memory effective i-vector extraction. IEEE/ACM Trans. Audio Speech Language Process. 22 (1), 248–259.

Dehak, N., Dehak, R., Kenny, P., Brummer, N., Ouellet, P., Dumouchel, P., 2009. Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification. In: Proc. Int. Speech Commun. Association (Interspeech), pp. 1559–1562.

Dehak, N., Dehak, R., Glass, J., Reynolds, D., Kenny, P., 2010. Cosine similarity scoring without score normalization techniques. In: Proc. IEEE Speaker and Language Recognition Workshop (Odyssey).

Dehak, N., Kenny, P., Dehak, R., Dumouchel, P., Ouellet, P., 2011. Front-end factor analysis for speaker verification. IEEE Trans. Audio Speech Language Process. 19 (4), 788–798.

Garcia-Romero, D., Espy-Wilson, C.Y., 2011. Analysis of i-vector length normalization in speaker recognition systems. In: INTERSPEECH, pp. 249–252.

Glembek, O., Burget, L., Matejka, P., Karafiát, M., Kenny, P., 2011. Simplification and optimization of i-vector extraction. In: Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), pp. 4516–4519.

Kenny, P., Boulianne, G., Dumouchel, P., 2005. Eigenvoice modeling with sparse training data. IEEE Trans. Speech Audio Process. 13 (3), 345–354.

Kenny, P., Boulianne, G., Ouellet, P., Dumouchel, P., 2007. Joint factor analysis versus eigenchannels in speaker recognition. IEEE Trans. Audio Speech Language Process. 15 (4), 1435–1447.

Kinnunen, T., Karpov, E., Fränti, P., 2004. Efficient online cohort selection method for speaker verification. In: Proc. Int. Conf. Spoken Language Process. (ICSLP), vol. 3, pp. 2401–2402.

Kinnunen, T., Karpov, E., Franti, P., 2006. Real-time speaker identification and verification. IEEE Trans. Audio, Speech, and Language Process. 14 (1), 277–288.

Martin, A.F., Greenberg, C.S., 2010. The nist 2010 speaker recognition evaluation. In: INTERSPEECH, pp. 2726–2729.

McClanahan, R.D., De Leon, P.L., 2012. Mixture component clustering for efficient speaker verification. In: Proc. Int. Speech Commun. Association (Interspeech).

McLaughlin, J, Reynolds, D.A., Gleason, T.P., 1999. A study of computation speed-ups of the GMM–UBM speaker recognition system. In: Proc. European Conf. Speech Communication and Technology (Eurospeech), pp. 1215–1218.

Mohammadi, H.R.S., Saeidi, R., 2006. Efficient implementation of gmm based speaker verification using sorted gaussian mixture model. Proc. Eur. Signal Process. Conf. (EUSIPCO), 4–8.

Reynolds, D., 2009. Gaussian mixture models. Encyclopedia of Biometrics. Springer.

Reynolds, D.A., Quatieri, T.F., Dunn, R.B., 2000. Speaker verification using adapted gaussian mixture models. In: Digital Signal Processing, p. 2000.

Runnalls, A., 2007. Kullback–Leibler approach to gaussian mixture reduction. IEEE Trans. Aerosp. Electron. Syst 43 (3), 989–999.

Saeidi, R., Kinnunen, T., Sadegh Mohammadi, H., Rodman, R., Fränti, P., 2010. Joint frame and gaussian selection for text independent speaker verification. In: Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), pp. 4530–4533.

Tydlitát, B., Navratil, J., Pelecanos, J., Ramaswamy, G., 2007. Text-independent speaker verification in embedded environments. In: Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), vol. 4, pp. 293–296.

Xiang, B., Berger, T., 2003. Efficient text-independent speaker verification with structural gaussian mixture models and neural network. IEEE Trans. Speech Audio Process. 11 (5), 447–456.

Ye, G., Mak, B., 2010. The use of subvector quantization and discrete densities for fast GMM computation for speaker verification. In: Proc. Int. Speech Commun. Association (Interspeech), pp. 1481–1484.

Ye, G., Mak, B., Mak, M., 2009. Fast GMM computation for speaker verification using scalar quantization and discrete densities. In: Proc. Int. Speech Commun. Association (Interspeech), pp. 2327–2330.