Voice Biometry Standard - Draft

Ondřej Glembek[1], Lukáš Burget[1], and Pavel Matějka[1]

[1]Speech@FIT group, Brno University of Technology, Czech Republic

July 17, 2015

Abstract

This document serves as a description of the proposed draft for a voice biometry standard.

1 Quick User Guide to i-vector Extraction

This standard is supposed to give formal description of the i-vector extraction algorithm. However, we provide a python demo package for i) better understanding of the properties and features of the extraction, and ii) for convenience, so that the user can immediately use the basic functions and do prompt customizations. For convenience, we start this document by a quick user guide to the i-vector extraction.

Python 2.7 was chosen as the implementation language. Apart from the standard python libraries, numpy and scipy modules are required for the package to work. The code is located in the `python` directory. The structure of the software is given by two main python executable scripts and a set of python modules. The two runnable scripts are:

- `raw2ivec.py` is the demo application for i-vector extraction. It directly implements this standard for easy usage. It takes six input command-line parameters:

  LIST_FILE
  List of files with relative paths and no extensions. This list defines the paths to the input and output files relative to the paths defined below. The audio files assume a `.wav` extension.

  VAD_DIR
  Directory with the voice-activity definition. If the parameter is set to `auto`, then VAD is computed using a predefined energy-based VAD algorithm. Otherwise, the files are assumed to be gzip-commpressed text file, whose format is two-column (space-separated) list of start

and end voiced-segments (in seconds). See `test/vad/fisher-english-p1` directory for examples. The extension of these files is expected to be `.lab.gz`.

WAV_DIR
Directory with the wave files. The files are expected to be in MS wave format and to have `.wav` extension.

UBM_DIR
The GMM UBM file. This is a gzipped text file where each line consist of a Gaussian weight, vector of its means, and a vector of the variances. This path should point to the `models/GMM.txt.gz` file, which is a part of this standard.

T_FILE
The i-vector extractor matrix. This is a gzipped text file with plain matrix definition. This path should point to the `models/v600_iter10.txt.gz` file, which is a part of this standard.

OUT_DIR
The output directory. The output files are saved to the relative paths as specified in the LIST_FILE, and their extension is set to `.i.gz`. The format is a gzipped text file, containing single line of the i-vector.

In the `test` directory, we provide the `testme_ivec.py.sh` one-touch script, which demoes the usage of the `raw2ivec.py` script on the example data.

- `runplda.py` is a demo for scoring the i-vectors. This script is NOT part of the standard and is provided as a courtesy to users who want to quickly test their i-vectors using a state-of-the-art PLDA backend.

ENROLL_LIST
List of enrollment files (i-vectors) with relative paths and no extensions. This list defines the paths to the input and output files relative to the paths defined below. The extension of these files is expected to be `.lab.gz`.

ENROLL_DIR
Directory with the enrollment i-vectors.

TEST_LIST
List of enrollment files (i-vectors) with relative paths and no extensions. This list defines the paths to the input and output files relative to the paths defined below. The extension of these files is expected to be `.lab.gz`.

TEST_DIR
Directory with the test i-vectors.

PLDA_MODEL_DIR
PLDA model definition directory. The directory contains the PLDA model matrices as well as other i-vector normalization parameters. This path should point to the `models/backend` directory.

OUT_FILE
The output directory. The output files are saved as a full matrix of scores where rows correspond to the individual enrollment i-vectors while columns correspond to the individual test i-vectors. The order is as specified by the corresponding lists.

2 Introduction

Human voice is an indispensable part of personal identity. The exchange of voice data in its raw format (waveform) is however complicated due to legislative issues in several countries around the world. The main objective of this initiative is to standardize voice representation based on *i-vectors* (to be defined later) as a common format for the exchange of voice data.

Multiple organizations have developed and deployed voice biometry technologies such as speaker verification systems, authentication systems, caller ID, etc. We believe that existence of such standard would facilitate the integration of their technologies into existing systems and seamless communication with other systems world-wide. The standard would define a simple, compact and easy-to-understand format of voice data that could be transmitted by all means of communication.

The proposed format for voice data is based on the i-vector paradigm. It preserves most of the statistical information needed for speaker verification without the need of recording the voice itself. Thus, it could be used in the same way as e.g. finger-prints in the forensic science today. This should help the police, intelligence, border service agencies as well as other commercial and non-commercial institutions to exchange information quickly and reliably without having to worry about legal issues.

2.1 What Is Voice Biometry

Voice biometry or speaker identification is the process of finding and attaching a speaker identity to the voice of an unknown speaker. Automated speaker identification systems do this by comparing voice with stored samples in a database of voice models. Voice biometrics alone can be used as a method of personal authentication. The information is extracted in the form of sufficient statistics as the person speaks. Instead of recording the whole utterance, the extracted statistics are compacted and subsequently used in the verification process.

Voice biometry can also be understood as an additional layer of security to the traditional password/passphrase systems. In this sense, a given password is pronounced by the person rather than being transmitted directly to the authenticating authority. Combined with encryption techniques, the voice-augmented password provides higher security for sensitive information.

Ever since their introduction in Speaker Recognition, i-vectors have been widely used in multiple fields of speech processing, such as Language Recognition [1], Age Estimation [2, 3], Emotion Detection [4], and even in Speech

Recognition [5, 6]. The so-called i-vector is an information-rich low-dimensional fixed-length vector extracted from the feature sequence representing a speech segment (see Section 3 for details on i-vector extraction).

Due to these properties, the i-vectors are ocassionally referred to as audio *voice-prints*[1]. As such, they can be used for audio indexing purposes, information exchange (e.g. forensic or intelligence agencies), speaker search, etc. Such usage, however, assumes that the i-vector extraction method (including the parameters of the method) is kept fixed, so that all i-vectors are compatible, and that their direct comparison is feasible.

2.2 The goals of this standard

The goals of this work are following:

- Define a unified i-vector extraction method, i.e. provide the algorithm description as well as the aglorithm parameter definition.
- Enable users of the voice biometry technologies (speaker verification, speaker identification, clustering of recordings based on speakers) to exchange voice biometry information without having to exchange audio recordings
- Have a reference software implementation that can be used for verification of new implementations
- Unify and open research of techniques that works with i-vectors (speaker comparison, speaker clustering, techniques for adaptation to news acoustic channels and conditions)
- Help the research and development (R&D) in speaker recognition by allowing the R&D labs to obtain i-vectors from their partners and customers who are reluctant to share raw audio for privacy and legal reasons.

The goal IS NOT to freeze any research and development of new algorithms and techniques. The idea is that the vendors would have the possibility to export/import standard i-vectors in their applications to enable data exchange. However, there is still expected to be a competition among vendors whose algorithms would keep up with the state-of-the-art research. It is expected that this standard be updated to next version in the future.

2.2.1 Level of Standardization

The following parts of the Speaker Recognition system are being standardized:

- The most wide spread algorithm (i-vectors) is standardized

[1] Let us stress that the term voice-print has been historically used in various ways, mainly in the forensics [7] and should therefore be treated with care [8, 9]. For this reason, we do not use this term through out the work. Rather, we conservatively keep using *i-vectors.*

- Only the minimal possible algorithmic part that enable data exchange is standardized, block that are not essential to get compatible results are free to change
- The data exchange formats is standardized
- The standard comes mainly from implementations and features that are spread among the research community

It is important to stress out that the following is not standardized:

- Voice activity detection
- i-vector post-processing
- i-vector scoring

All of these steps will be described in the following sections.

3 The Definition of the Standard

The i-vector systems have become the state-of-the-art technique in the speaker verification field [10]. They provide an elegant way of reducing the large-dimensional input data to a small-dimensional feature vector while retaining most of the relevant information. The technique was originally inspired by Joint Factor Analysis framework introduced by Patrick Kenny [11, 12].

Let us first state the motivation for the i-vectors. The main idea is that the speaker- and channel-dependent GMM supervector $\mathbf{s}$ can be modeled as:

$$\mathbf{s} = \mathbf{m} + \mathbf{T}\mathbf{w} \tag{1}$$

where $\mathbf{m}$ is the UBM GMM mean supervector, $\mathbf{T}$ is a low-rank matrix representing M bases of the reduced total variability space, and $\mathbf{w}$ is a standard normal distributed vector of size M.

For each observation $\mathcal{X}$, the aim is to estimate the parameters of the posterior probability of $\mathbf{w}$:

$$\mathrm{p}(\mathbf{w}|\mathcal{X}) = \mathcal{N}(\mathbf{w}; \mathbf{w}_{\mathcal{X}}, \mathbf{L}_{\mathcal{X}}^{-1}) \tag{2}$$

The i-vector is the MAP point estimate of the variable $\mathbf{w}$, i.e. the mean $\mathbf{w}_{\mathcal{X}}$ of the posterior distribution $\mathrm{p}(\mathbf{w}|\mathcal{X})$. It maps most of the relevant information from a variable-length observation $\mathcal{X}$ to a fixed- (small-) dimensional vector. $\mathbf{T}$ is reffered to as the i-vector extractor.

This standard defines the algorithm of extracting the i-vector from the input data, which are in the form of an audio recording.

To deploy the full speaker recognition system, the user needs a classifier for i-vector comparison. This stage is not part of this standard. However, as a courtesy and for sake of completness, we are including a description and a demo application of a commong classifier—the PLDA. The system diagram is shown in Fig. 1.

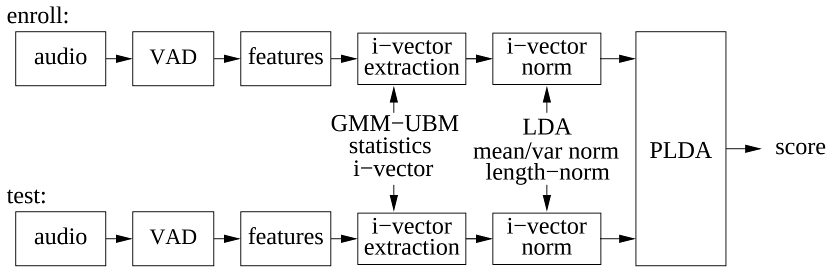

Figure 1: Overall schematic of the system.

Let us stress out that the system has been optimized for telephone audio quality. This is reflected in the feature extraction as well as in the data which were used to train the different steps of the i-vector extraction.

Let us now describe each stage of the system in detail.

3.1 Input Data

Feature Extraction—MFCC

First, preemphasis is performed on the signal by using a high-pass filter, which is defined as a difference of the current audio sample and a scaled-down previous audio sample. The scale constant is referred to as the *preemphasis coefficient.*

We use cepstral features, extracted using a 25 ms Hamming window. We use 24 Mel-filter banks and we limited the bandwidth to the 125–3800Hz range. 19 Mel frequency cepstral coefficients together with zero-*th* coefficient are calculated every 10 ms.

This 20-dimensional feature vector is subjected to short time mean- and variance-normalization using a 3 s sliding window. Delta and double delta coefficients were then calculated using a five-frame window giving a 60-dimensional feature vector.

The Code

The code for the MFCC extraction is located in the `features.py` module. At the start of the main script, the filter-bank definition is computed using the `mel_fbank_mx` function. The `winlen_nfft` parameter defines the analysis window length, set to 200 audio samples. `fs` is the sampling frequency of the audio (constantly set to 8000 Hz). `NUMCHANS` defines the 24 Mel-filter banks, and `LOFREQ` and `HIFREQ` parameters define the lower 125 Hz and upper 3800 Hz filter-bank range, respectively.

Once the Mel filter-banks is defined, the core MFCC computation is performed using the `mfcc_htk` function. The following paremeters are specified:

- `sig` — array of the acoustic signal in 8000 Hz sampling frequency

- `window` — framing window length, set to 200 acoustic samples (25 miliseconds windows width)
- `noverlap` — window overlap set to 120 acoustic samples (10 miliseconds window shift)
- `fbank_mx` — filter-bank definition as defined in the previous paragraph
- `_0` — indicates at which position the zero-th MFCC coefficient apperas (set to "first" to prepend the coefficient to the feature vector)
- `NUMCEPS` — number of required cepstrum coefficients, set to 19
- `RAWENERGY` — includes the frame Energy, set to True
- `PREEMCOEF` — defines the preemphasis coefficient, set to 0.97
- `CEPLIFTER` — balancing coefficient of the cepstral coefficients by a window for a flatter dynamic range, set to 22
- `ZMEANSOURCE` — removes the DC offset locally from each frame, set to True
- `ENORMALISE` — normalizes the energy
- `ESCALE` — energy scaling constant, set to 0.1
- `SILFLOOR` — energy flooring constant, set to 50
- `USEHAMMING` — set to true to weight a signal frame by a Hamming window function

The feature extraction is augmented by adding first- and second-order derivatives. This step is marked as `[add_deriv]` and is computed in the `add_deriv` function. The derivatives are computed using a 5-frame window—2 frames left context, 2 frames right context, and the current frame.

In the the following step (`[reshape]`), the feature coefficients are re-ordered to match the model training. This step has no functional meaning, however the training procedure (which is not included in this work) defined the feature order in a different way.

Sufficient Statistics

The input data for the observation $\mathcal{X}$ is given as a set of *zero-* and *first-order statistics* — $\mathbf{n}_{\mathcal{X}}$ and $\mathbf{f}_{\mathcal{X}}$. These are extracted from F dimensional features using a GMM UBM with C mixture components, defined by a mean supervector $\mathbf{m}$, block-diagonal covariance matrix $\mathbf{\Sigma}$, and a vector of mixture weights ω. For each Gaussian component c, the statistics are given respectively as:

$$N_{\mathcal{X}}^{(c)} = \sum_t \gamma_t^{(c)} \quad (3)$$

$$\mathbf{f}_{\mathcal{X}}^{(c)} = \sum_t \gamma_t^{(c)} \mathbf{o}_t \quad (4)$$

Table 1: *GMM UBM training corpus statistics.*

Database	Amount of speech data [hours]
nist-sre-test2004	35.80
nist-sre-train2004	97.32
nist-sre-test2005	59.22
nist-sre-train2005	95.00
nist-sre-test2006	82.59
nist-sre-train2006	64.73
nist-sre-dev2008	43.15
nist-sre-follow2008	142.54
nist-sre-test2008	199.99
nist-sre-train2008	335.64
Total	1156.03

where $\mathbf{o}_t$ is the corresponding feature vector in time t, and $\gamma_t^{(c)}$ is its occupation probability. The complete zero- and first-order statistics supervectors are $\mathbf{f}_{\mathcal{X}} = \left(\mathbf{f}_{\mathcal{X}}^{(1)'}, \ldots, \mathbf{f}_{\mathcal{X}}^{(C)'}\right)'$, and $\mathbf{n}_{\mathcal{X}} = \left(N_{\mathcal{X}}^{(1)}, \ldots, N_{\mathcal{X}}^{(C)}\right)'$.

The GMM UBM parameters are part of this standard. They were trained on the NIST SRE 2004-2008 data. The GMM UBM contains 2048 Gaussian components and the model was trained using the EM algorithm by sequential doubling the number of Gaussians 10-iteration EM cycles. Both training and test sets of the corpora were used. The amount of data used is summarized by Tab. 1.

The Code

The GMM statistics are computed via the `gmm_eval` function defined in the `gmm` module. The function takes two inputs: the matrix of input data and the GMM definition. The algorithm is optimized for speed where quadratic data expansion is involved, therefore, it has quadratic memory complexity w.r.t. number of data points. Therefore, the statistics are computed in batches of 1000 samples and summed in a loop.

Statistics Normalization

For convenience, we *center* the first order statistics around the UBM means, which allows us to treat the UBM means effectively as a vector of zeros:

$$\begin{aligned} \mathbf{f}_{\mathcal{X}}^{(c)} &\leftarrow \mathbf{f}_{\mathcal{X}}^{(c)} - N_{\mathcal{X}}^{(c)}\mathbf{m}^{(c)} \\ \mathbf{m}^{(c)} &\leftarrow \mathbf{0} \end{aligned}$$

Similarily, we "normalize" the first-order statistics and the matrix $\mathbf{T}$ by the UBM covaricances, which again allows us to treat the UBM covariances as an identity matrix[2]:

$$\begin{array}{rcl} \mathbf{f}_{\mathcal{X}}^{(c)} & \leftarrow & \mathbf{\Sigma}^{(c)-\frac{1}{2}}\mathbf{f}_{\mathcal{X}}^{(c)} \\ \mathbf{T}^{(c)} & \leftarrow & \mathbf{\Sigma}^{(c)-\frac{1}{2}}\mathbf{T}^{(c)} \\ \mathbf{\Sigma}^{(c)} & \leftarrow & \mathbf{I} \end{array}$$

where $\mathbf{\Sigma}^{(c)-\frac{1}{2}}$ is a Cholesky decomposition of an inverse of $\mathbf{\Sigma}^{(c)}$, and $\mathbf{T}^{(c)}$ is an $F \times M$ sub-matrix of $\mathbf{T}$ corresponding to the c mixture component such that $\mathbf{T} = \left(\mathbf{T}^{(1)'}, \ldots, \mathbf{T}^{(C)'}\right)'$.

The Code

The statistics normalization is computed in the `normalize_stats` function defined in the main script `raw2ivec.py`. It takes four parameters as an input (vectors of zero-order statistics `n`, vector of first-order statistics `f`, super-vector of ubm gmm means `ubm_means`, and a super-vector of the inverted UBM GMM standard deviations `ubm_norm`). The output is a tuple of normalized zero- and first-order statistics.

3.2 i-vector Extraction

As described in [11] and with the data transforms from previous section, for an observation $\mathcal{X}$, the corresponding i-vector is computed as a point estimate:

$$\mathbf{w}_{\mathcal{X}} = \mathbf{L}_{\mathcal{X}}^{-1}\mathbf{T}'\mathbf{f}_{\mathcal{X}} \tag{5}$$

where $\mathbf{L}$ is the precision matrix of the posterior distribution, computed as:

$$\mathbf{L}_{\mathcal{X}} = \mathbf{I} + \sum_{c=1}^{C} N_{\mathcal{X}}^{(c)}\mathbf{T}^{(c)'}\mathbf{T}^{(c)} \tag{6}$$

The computational complexity of the whole estimation for one observation is $O(CFM + CM^2 + M^3)$. The first term represents the $\mathbf{T}'\mathbf{f}_{\mathcal{X}}$ multiplication. The second term represents the sum in (6) and includes the multiplication of $\mathbf{L}_{\mathcal{X}}^{-1}$ with a vector. The third term represents the matrix inversion.

The memory complexity of the estimation is $O(CFM + CM^2)$. The first term represents the storage of all the input variables in (5), and the second term represents the precomputed covarance matrices in the sum of (6).

Note that the computation complexity grows quadratically with M in the sum of (6), and linearly with C. This becomes the bottle-neck in the i-vector computation, resulting in high memory and CPU demands.

[2]Part of the factor estimation is a computation of $\mathbf{T}'\mathbf{\Sigma}^{-1}\mathbf{f}$, where the decomposed $\mathbf{\Sigma}^{-1}$ can be projected to the neigboring terms, see [11] for the detailed formulas.

The Code

The i-vector extraction is implemented in the `ivector.py` module. The $\mathbf{T}^{(c)\prime}\mathbf{T}^{(c)}$ terms from (6) for all Gaussian components (c) are precomputed using the function `compute_VtV` at the begining of the program, since we only need to compute them once per batch. Eq. (5) is implemented in the `estimate_i` function.

3.3 i-vector Binary and ASCII Formats

The i-vectors could essentially be stored in any format, however, we chose to introduce a binary format that—appart from the i-vector itself—allows to store the amount of speech that was used for the i-vector extraction, metadata, and a CRC32 checksum. The ASCII version is a Base64 version of the binary byte array [13].

Let us now describe the format itself by listing the fields as stored in the file. All numeric values are stored as little-endian.

ID string
: 4-byte id string containing "VBS1"

Version
: 4-byte signed integer defining the version of the standard (set to 1 in this release)

Audio length
: 4-byte float storing number of seconds that were used for the i-vector extraction. Note that this field is mandatory as it is typically used in system calibration.

i-vector dimensionality
: 4-byte signed integer storing number of i-vector dimensions. This is set to 600 in this release.

i-vector definition
: Array of 4-byte float numbers. The length of this array is defined by the i-vector dimensionality field.

Metadata length
: 4-byte signed integer defining the size (in bytes) the metadata field. The metadata is suggested to be pairs of null-terminated strings, each pair being a key–value record in ASCII representation.

Metadata
: Array of chars whose size is defined by the "Metadata length" value. The metadata is suggested to be pairs of null-terminated strings, each pair being a key–value record in ASCII representation.

CRC32 checksum
: 4-byte signed integer defining the CRC32 checksum of all previous fields.

The Code

The python code for reading and writing the code is located in the `ivector_io.py` module. Reading the ivector is performed via the `read_binary_ivector` function whose input is the filename and the function returns a 3-tuple of the i-vector, audio length, and the metadata field (set to None if no metadata is stored). Writing is implemented in the `write_binary_ivector` function, whose input is the i-vector, the audio length (in seconds), and an optional field of metadata.

3.4 i-vector Extractor Training

Although not part of the standard, let us briefly describe the procedure by which the i-vector extractor was trained. Model hyper-parameters $\mathbf{T}$ are estimated using the same EM algorithm as in case of JFA [11]. Our algorithm makes use of an additional *minimum divergence* update step [12, 14].

In the E step, the following accumulators are collected for all training observations i:

$$\mathbf{C} = \sum_i \mathbf{f}_i \mathbf{w}_i' \tag{7}$$

$$\mathbf{A}^{(c)} = \sum_i N_i^{(c)} \left(\mathbf{L}_i^{-1} + \mathbf{w}_i \mathbf{w}_i' \right) \tag{8}$$

where $\mathbf{w}_i$ and $\mathbf{L}_i$ are the estimates from (5) and (6) for observation i. The M step update is given as follows:

$$\mathbf{T}^{(c)} = \mathbf{C}\mathbf{A}^{(c)^{-1}} \tag{9}$$

The data used for computing the i-vector extractor matrix $\mathbf{T}$

The amount of i-vector extractor training data is summarized by Tab. 2. The training corpus consists of the following databases: Fisher English (part 1 and 2), NIST SRE 2004–2008, Switchboard (phase 2, phase 3, celluar part 1, and celluar part 2).

3.5 Scoring

The scoring procedure is not intended to be the part of the standard. However for convenience, we are providing an example that is being used in the latest Speaker Recognition systems. First, we pre-process the i-vectors, and then, we use a binary classifier to compute a likelihood-ratio of the same- and different-speaker hypothesis for a given trial, i.e. an i-vector pair.

3.5.1 i-vector Pre-Processing

Before we apply the linear classifier to get a score of a trial, the i-vectors are pre-processed. The same technique as in [10, 15] is used in this work. The extracted i-vectors are scaled down using an LDA matrix to 200 dimensions, and

Table 2: *I-vector extractor training corpus statistics.*

Database	Amount of speech data [hours]
fisher-english-p1	1954.84
fisher-english-p2	1936.19
nist-sre-all2010/interview/3min	37.44
nist-sre-all2010/interview/8min	57.74
nist-sre-all2010/phonecall/mic	18.79
nist-sre-all2010/phonecall/tel	69.20
nist-sre-test2004	99.51
nist-sre-train2004	286.84
nist-sre-test2005	388.16
nist-sre-train2005	272.66
nist-sre-test2006	448.68
nist-sre-train2006	185.48
nist-sre-dev2008	61.76
nist-sre-test2008	375.14
nist-sre-train2008	763.34
nist-sre-follow2008	217.92
sw2_phase2	740.19
sw2_phase3	439.59
sw_cellular_part1	258.99
sw_cellular_part2	397.69
Total	9010.23

further normalized by a within-class covariance matrix. Both of these matrices are gender-independent and were estimated on the same data as the i-vector extractor, except the Fisher data was excluded, resulting in 1684 female speakers in 715 hours of speech and 1270 male speakers in 537 hours of speech.

The Code

The normalization parameters are stored in the `models/backend` directory as `backend.LDA.txt.gz` for combined LDA projection and variance normalization, and `backend.mu_train.txt.gz` for global mean removal.

3.5.2 PLDA

To compare of i-vectors in a verification trial, we use a Probabilistic Linear Discriminant Analysis (PLDA) model [16, 17]. It can be seen as a special case of JFA with a single Gaussian component. Given a pair of i-vectors, PLDA allows to compute the log-likelihood for the same-speaker hypothesis and for the different-speaker hypothesis. One can directly evaluate the log-likelihood ratio of the same-speaker and different-speaker trial using

$$\begin{aligned} s(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2) &= \boldsymbol{\phi}_1^T \boldsymbol{\Lambda} \boldsymbol{\phi}_2 + \boldsymbol{\phi}_2^T \boldsymbol{\Lambda} \boldsymbol{\phi}_1 + \boldsymbol{\phi}_1^T \boldsymbol{\Gamma} \boldsymbol{\phi}_1 + \boldsymbol{\phi}_2^T \boldsymbol{\Gamma} \boldsymbol{\phi}_2 \\ &+ (\boldsymbol{\phi}_1 + \boldsymbol{\phi}_2)^T \mathbf{c} + k, \end{aligned} \tag{10}$$

where $\boldsymbol{\Lambda}$, $\boldsymbol{\Gamma}$, $\mathbf{c}$, k are derived from the parameters of PLDA as in [18].

The Code

The PLDA code as defined in Eq. (10) is implemented in the `bilinear_plda` function directly in the `runplda.py` script. The function takes the PLDA parameters as an input as well as matrices of input test and enroll i-vectors. Full matrix of score (all i-vectors from test are scored with all i-vectors from enroll) is then returned as an output. Running the script from the command line has been described in Sec. 1.

4 Final Remarks

- This standard (and especially parameters of models) was developed and tested on telephone data on which it should provide faitr performance. Please, do not expect it to perform well on far-field, noisy, multi-channel or cross-talked data.
- VAD performance is crucial for any spekaer recognition algorithms. The provided energy-based VAD is a rudimentary one. The best you can do is to check on your data with our implementation, then replace the VAD for a better one.
- the code is not optimized for speed and or memory usage.
- when re-implementing the standard in other languages and/or other architectures and operating systems, please always check that the produced results are the same (not bit-exact but if the difference of the same i-vector element is 1% or more, there is a problem)

5 Contacts

- Voice Biometry Standard's web page: `http://voicebiometry.org/`
- VBS's email: `mailto:info@voicebiometry.org/`
- Google group: `http://groups.google.com/d/forum/voice-biometry-standard` and `mailto:voice-biometry-standard@googlegroups.com`

References

[1] David González Martínez, Oldřich Plchot, Lukáš Burget, Ondřej Glembek, and Pavel Matějka, “Language recognition in ivectors space,” in *Proceedings of Interspeech 2011*. 2011, vol. 2011, pp. 861–864, International Speech Communication Association.

[2] Mohamad Hasan Bahari, Mitchell McLaren, Hugo Van hamme, and David A. van Leeuwen, “Speaker age estimation using i-vectors,” *Eng. Appl. of AI*, vol. 34, pp. 99–108, 2014.

[3] Anna Fedorova, Ondrej Glembek, Pavel Matejka, and Tomi Kinnunen, “Exploring ANN back-ends for i-vector based speaker age estimation,” in *Submitted to Interspeech, 2015*, 2015.

[4] Marcel Kockmann, Lukáš Burget, and Jan Černocký, “Application of speaker- and language identification state-of-the-art techniques for emotion recognition,” *Speech Communication*, vol. 53, no. 9, pp. 1172–1185, 2011.

[5] Martin Karafiát, Lukáš Burget, Pavel Matějka, Ondřej Glembek, and Jan Černocký, “ivector-based discriminative adaptation for automatic speech recognition,” in *Proceedings of ASRU 2011*. 2011, pp. 152–157, IEEE Signal Processing Society.

[6] G. Saon, H. Soltau, D. Nahamoo, and M. Picheny, “Speaker adaptation of neural network acoustic models using i-vectors,” in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, Dec 2013, pp. 55–59.

[7] Louis-Jean Boë, “Forensic voice identification in france,” *Speech Communication*, vol. 31, no. 2̆20133, pp. 205 – 224, 2000.

[8] Jean-François Bonastre, Louis-Jean Bimbot, Frédéric an Boë, Joseph P. Campbell, Douglas A. Reynolds, and Ivan Magrin-Chagnolleau, “Person authentication by voice: a need for caution.,” in *INTERSPEECH*. 2003, ISCA.

[9] J.P. Campbell, W. Shen, W.M. Campbell, R. Schwartz, J.-F. Bonastre, and D. Matrouf, “Forensic speaker recognition,” *Signal Processing Magazine, IEEE*, vol. 26, no. 2, pp. 95–103, March 2009.

[10] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, “Front-end factor analysis for speaker verification,” *IEEE Transactions on Audio, Speech and Language Processing*, vol. PP, no. 99, pp. 1 –1, 2010.

[11] P. Kenny, “Joint factor analysis of speaker and session variability : Theory and algorithms - technical report CRIM-06/08-13. Montreal, CRIM, 2005,” 2005.

[12] P. Kenny, G. Boulianne, P. Oullet, and P. Dumouchel, "Joint factor analysis versus eigenchannels in speaker recognition," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, no. 7, pp. 2072–2084, 2007.

[13] "Base64 Wikipedia article," https://en.wikipedia.org/wiki/Base64.

[14] Niko Brümmer, "The em algorithm and minimum divergence," Agnitio Labs Technical Report. Online: http://niko.brummer.googlepages.com/EMandMINDIV.pdf, Oct. 2009.

[15] Daniel Garcia-Romero, "Analysis of i-vector length normalization in Gaussian-PLDA speaker recognition systems," 2011, Submitted to ICSLP 2011.

[16] S. J. D. Prince and J. H. Elder, "Probabilistic linear discriminant analysis for inferences about identity," in *11th International Conference on Computer Vision*, 2007, pp. 1–8.

[17] Patrick Kenny, "Bayesian speaker verification with heavy–tailed priors," in *Proc. of Odyssey 2010*, Brno, Czech Republic, June 2010, http://www.crim.ca/perso/patrick.kenny, keynote presentation.

[18] L. Burget, O. Plchot, S. Cumani, O. Glembek, P. Matějka, and N. Brümmer, "Discriminatively trained probabilistic linear discriminant analysis for speaker verification," in *Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Prague, CZ, May 2011, accepted for publication.