1 Abstract

This document serves to explain the steps involved in building a speaker verification system based on the Total Variability approach. Currently, the use of Total Variability has achieved state of the art performance in the NIST SRE tasks; furthermore, the use of a fast cosine scoring metric makes this approach scalable to real-world applications. This tutorial will merely outline the basic steps involved in building a speaker verification system. A lot of the rigor and theory will only be referenced. As this is very much just a work in progress, please feel free to send your feedback to sshum @ mit . edu. Thanks!

2 Introduction

In recent years, factor analysis-based approaches have achieved the state of the art for text-independent speaker detection tasks. In an effort to enhance the classical method of modeling speakers using Gaussian Mixture Models (GMMs) [1], methods developed in Joint Factor Analysis (JFA) present powerful tools to better represent speaker variabilities and compensate for channel and, more generally, session inconsistencies [2]. Recently, a factor analysis-based approach to speaker recognition using just a cosine similarity metric between low-dimensional vectors proved highly effective [3]. Unlike traditional JFA, this “Total Variability Approach” avoids the joint estimation of separate speaker and session spaces and factors; instead, factor analysis is first used to define a low-dimensional subspace that contains both speaker and session variabilities, then, in that space, the inter-session variabilities are subsequently removed [4].

More specifically, classical JFA modeling defines respective subspaces for the speaker and the channel factors, then estimates them jointly [3]. A more recent approach represents all the factors in a (single) total variability space with no distinction made between speaker and session subspaces [4]. The speaker- and session-dependent supervector\(^1\) \(M\) is defined as

\[
M = m + Tw
\]

where \(m\) is the speaker- and session-independent supervector commonly taken from a Universal Background Model (UBM)\(^2\), \(T\) is a rectangular matrix of low rank that defines the total variability space, and \(w\) is a random vector with a normally distributed prior \(\mathcal{N}(0, I)\). The components of \(w\) are referred to as the “total factors”, and \(w\) will be referred to as a “total factor vector.”

This method was motivated by the realization that session/channel factors estimated in JFA also contained information about the speaker. As such, instead of working in the high-dimensional supervector space, what was proposed was a way to capture, in a single subspace, the “Total” variability of both speaker and session subspaces before subsequently removing the unwanted inter-session variabilities.

\(^1\)A supervector is composed by stacking the mean vectors from a GMM.

\(^2\)A UBM is a large GMM trained to represent the speaker-independent distribution of features [1].
3 Overview

The main steps for building this system are as follows:

1. Speech Activity Detection and Feature Extraction (including Feature Warping / Gaussianization)
2. UBM Building
3. Total Variability Matrix Training
4. Total Factors Extraction
5. Inter-session Compensation
6. Cosine Scoring
7. Score Normalization

As this is a mere introduction to the project, we will only discuss those items that have not been italicized. While we would be happy to talk about these ideas or provide references for such topics, these omitted sections are a bit advanced and so the relevant exercises have been left out to preserve the overall picture we are trying to present.

3.1 Prerequisites

The remainder of this lab assumes that you are familiar with the basics of speech technology. That is, we are assuming that you understand all the ideas behind front-end feature extraction (i.e. MFCCs) as well as the basics of Gaussian Mixture Models (GMMs) and their training. We will skip the mathematics, etc. behind these ideas, as it is assumed to be thoroughly understood.

4 Data

The data we are using for this task is from the National Institute of Standards and Technology (NIST) Speaker Recognition Evaluation (SRE) of 2004. In the current version of this tutorial, we only have data for training. When things become more developed, we hope to also have a self-contained test set on which users can run experiments and match some of the baseline results we produce so as to ascertain their understanding of this technology. For now, however, we will stick to a less constrained testing environment.

5 Feature Extraction

We typically build a speaker verification system out of cepstral features extracted every 10ms using a 25ms Hamming window, following a pretty standard configuration for most speech-related applications. In our case, we extract 19 mel-frequency cepstral coefficients
(MFCCs) along with the log energy. This gives us a 20-dimensional feature vector, which is then subject to feature warping [5] using a sliding window three seconds in length. From here, delta (Δ) and delta-delta (ΔΔ) coefficients are calculated every five frames to finally produce a 60-dimensional feature vector.

5.1 Laboratory Tutorial

You should have access to the directory /data/scratch/najim/LAB/, under which you will find subdirectories BG, MFCC, SEG, and script, as well as a file list_audio_file.lst. The script located at /data/scratch/najim/LAB/script/feature_extraction.pl contains all the necessary tools for extracting the relevant features from a list of audio files.

Running the script “feature_extraction.pl” will dump a bunch of files of MFCCs in the MFCC directory. To see the result, run HList /data/scratch/najim/LAB/MFCC/<filename> \^3, where <filename> is the name of some file in that directory.

These features were extracted using HTK (http://htk.eng.cam.ac.uk/). The details of the code and all do not matter; however, you will find in your script directory two .config files that are used by HTK to extract features. One of these files was used before the aforementioned feature warping step, while the other was used after. Study these two files briefly and, at the very least, be able to comment on which .config file is used first and why.

6 UBM Building

We proceed with our modeling of speech with the building of a Universal Background Model (UBM). The UBM is a large Gaussian Mixture Model (GMM) trained to represent the speaker-independent distribution of features; in particular, we are looking for speech that is reflective of the expected speech to be encountered during recognition. This applies to the type of speech (i.e. casual conversation or business meeting or telephone service), the quality and channel (i.e. close-talking microphone? microphone array? shouting outdoors? telephone?), and the speaker population (i.e. males, females and/or children). A good amount of care is usually involved in selecting the data used in building this UBM. In our simple case, however, we will be using data from the NIST Speaker Recognition Evaluations to train our model.

The parameters of our UBM are trained using the Expectation-Maximization algorithm [6]. That is, given a collection of training vectors (the MFCCs extracted every 10ms from the previous step), we can obtain the maximum likelihood (ML) parameters of our UBM via the EM algorithm. The EM algorithm iteratively refines the GMM parameters to monotonically increase the likelihood of the estimated model for the observed feature vectors [1]. The equations for these ML parameter updates for each Gaussian i, where i ∈ {1, ..., C} can be found in [4].

In deciding the number of Gaussians to use for the UBM, there are a number of ways to do so. However, that is a bit beyond the scope of this work, so we will simply specify the use

\(^3\)You will need to make sure that /usr/local/htk/install/bin is included into your $PATH
of 256 Gaussians for the building of the UBM. This should allow the training to complete in a reasonable amount of time.

6.1 Laboratory Tutorial

Running the script in /data/scratch/najim/LAB/script/runUBM.sh should build a UBM based on the features that were extracted in the previous step. Take a look at the script and pay some attention to the options labeled `--MinIterEM` and `--MaxIterEM`. These stand for “Minimum Iterations of EM” and “Maximum Iterations of EM”, respectively. Why might it be necessary to specify the minimum and maximum iterations of running the EM algorithm? In particular, what might happen if we did not specify a limit to the maximum number of EM iterations? You might find it helpful to ask around or to look into the references that have been cited.

7 Total Variability

In this section we are looking to train the total variability matrix $T$. Recall that, in factor analysis-based approaches, what we have here is a speaker- and session-dependent supervector $M$ defined as

$$M = m + Tw$$

where $m$ is the speaker- and session-independent supervector commonly taken from a Universal Background Model (UBM), $T$ is a rectangular matrix of low rank that defines the total variability space, and $w$ is a random vector with a normally distributed prior $\mathcal{N}(0, I)$.

The matrix $T$ can be trained following a similar process to that of learning the eigenvoice matrix of JFA, which is more fully detailed in [4]. The main difference between the two is that in training the eigenvoice of JFA, all recordings of a given speaker are considered to belong to the same person, whereas in training $T$, each instance of a given speaker’s set of utterances is regarded as having been produced by a different speaker.

From here, most of the work we do will require, for a given utterance, the Baum-Welch statistics from the UBM [3]. Suppose our given utterance $u$ is represented as a sequence of $L$ frames $u = \{y_1, y_2, ..., y_L\}$. Then the relevant Baum-Welch statistics are

$$N_c(u) = \sum_{t=1}^{L} P(c|y_t, \Omega)$$

$$F_c(u) = \sum_{t=1}^{L} P(c|y_t, \Omega) y_t$$

where $c = 1, ..., C$ is the index of the corresponding Gaussian component and $P(c|y_t, \Omega)$ corresponds to the posterior probability of generating the frame $y_t$ by mixture component $c$. Now for our purposes, we define the centralized first order Baum-Welch statistics based on
the mean of the mixture components in the UBM:

\[
\tilde{F}_c(u) = F_c(u) - N_c(u)m_c
\]

\[
= \sum_{t=1}^{L} P(c|y_t, \Omega) (y_t - m_c)
\]

where \(m_c\) is the mean of mixture component \(c\).

These Baum-Welch statistics will also be relevant in the next section, when we discuss the extraction of the i-vectors.

### 7.1 Laboratory Tutorial

The script `/data/scratch/najim/LAB/script/doBaumWelch.sh` will extract the Baum-Welch statistics from all pertinent files in the list of given audio files. Afterwards, the script `runJFA.sh` in the same script directory contains the command used to train the total variability matrix \(T\). The rank of this matrix is chosen by the creators of this system; it is usually a function of the application, the training or testing data, and a number of other extraneous factors. To spare you the need to do so, we have chosen a rank of 200 for this tutorial, though it has ranged from 200 to 600 in past experience.

### 8 Total Factors Extraction

In our approach, we use factor analysis as a front-end to extract speaker-relevant features from a given utterance. What this comes down to is the reduction of every input (speech utterance) into a vector \(w\), which, as we have seen previously, is of dimension equal to the rank of our total variability matrix \(T\). We assume that \(w\), our “total factors vector” (also known as “i-Vectors” in other camps) is a random vector with a normally distributed prior \(\mathcal{N}(0, I)\). Given the centralized first order Baum-Welch statistics as discussed in the previous section, we obtain the total factors vector for utterance \(u\) via the following:

\[
w = (I + T^t\Sigma^{-1}N(u)T)^{-1} \cdot T^t\Sigma^{-1}\tilde{F}(u)
\]

where \(N(u)\) is the diagonal matrix of dimension \(CF \times CF\) whose diagonal blocks are \(N_c(u)I\), \((C = 1, \ldots, C)\) and \(\tilde{F}(u)\) is a supervector of dimension \(CF \times 1\) obtained by concatenating all the centralized first order Baum-Welch statistics \(\tilde{F}_c(u)\). Here, \(\Sigma\) is a diagonal covariance matrix of dimension \(CF \times CF\) that is estimated during the training of \(T\). It models the residual variabilities not captured by the total variability matrix \(T\) [7].

### 8.1 Laboratory Tutorial

In an effort to be concise, we have spared much of the mathematical details. The script `/data/scratch/najim/LAB/script/computeW.sh` contains the commands to extract the total factors vectors from all the utterances in our list of files. These vectors should appear in `.txt` format in the directory `/data/scratch/najim/LAB/TotalFactorW/`. 
9 Cosine Scoring

Given an utterance $Y$ and a hypothesized speaker $S$, the task of speaker verification is to determine if $Y$ was spoken by $S$. To keep things simple, we usually make the assumption that $Y$ contains speech from only one speaker. This single-speaker detection task can be restated as a basic hypothesis test between $H_1$, where $Y$ is from the hypothesized speaker $S$, and $H_0$, where $Y$ is not from the hypothesized speaker [1]. To make this decision, we apply the likelihood ratio test given by

$$\frac{p(Y|H_1)}{p(Y|H_0)} \begin{cases} \geq \theta & \text{accept } H_1 \\ < \theta & \text{reject } H_1 \end{cases} \quad (8)$$

The whole point of speaker verification is to determine techniques to compute values for the two likelihoods, $p(Y|H_0)$ and $p(Y|H_1)$ [1].

In terms of nomenclature, this likelihood ratio (or log likelihood ratio) can be seen as a “score” that is compared against a given decision threshold $\theta$. In the traditional GMM-UBM approach, the actual values of the ratio can be computed; likewise, we can also do so in JFA. However, there has been a good amount of recent work to simplify and speed up the computation in factor analysis-based approaches; as such, experiments were run on ways to approximate the log likelihood ratio.

Much of the success of the Total Variability Approach hinges on the speed and simplicity of the cosine similarity metric [3]. Given two total factor vectors generated by (7) via the projection of two supervectors in the total variability space, a target $w_{\text{target}}$ from a known speaker and a test $w_{\text{test}}$ from an unknown speaker, the cosine similarity score is given as

$$\text{score}(w_{\text{target}}, w_{\text{test}}) = \frac{(w_{\text{target}})^t (w_{\text{test}})}{\|w_{\text{target}}\| \|w_{\text{test}}\|} \quad (9)$$

$$= \frac{(w'_{\text{target}})^t (w'_{\text{test}})}{\|w'_{\text{target}}\| \|w'_{\text{test}}\|} \quad (10) \geq \theta \quad (11)$$

where $w' = \frac{w}{\|w\|}$ is the length-normalized version of $w$, and $\theta$ is the decision threshold. We can see that (10) is the simple dot product and that this scoring function is considerably less complex than the log likelihood ratio (LLR) scoring operations used in JFA and other methods [8].

9.1 Laboratory Tutorial

Up until now, we have provided all the scripts that were necessary to build this system. This last section we will leave up to you. You should have realized that all the Total Factor Vectors extracted in the previous section come in text format, so it should not be very difficult to implement the cosine scoring function on your own.

10 Conclusion

This tutorial has hopefully provided some insight into the basics of how a speaker verification system is build. We have done so via the Total Variability Approach and discussed
the following steps: Feature Extraction, UBM Building, Total Variability Matrix Training, Total Factors (i-vector) Extraction, and Cosine Scoring. Indeed, we have glossed over some mathematical details as well as skipping some parts such as Inter-session Compensation and Score Normalization. However, those can be found in and should be easily implemented based on the provided references.

References


