SPEAKER VERIFICATION: A FLEXIBLE PLATFORM ARCHITECTURE FOR EXPERIMENT DESIGN AND EXECUTION

Jorge Prendes, Marc S. Ressl, Roxana Saint-Nom

Grupo de electrónica digital aplicada, ITBA (Instituto Tecnológico de Buenos Aires)

Av. E. Madero 399, Buenos Aires, Argentina

{jprendes,mressl,saintnom}@itba.edu.ar

ABSTRACT
A new speaker verification platform architecture is introduced. Its design is focused on flexibility and ease of use, allowing for rapid generation of diverse experimental configurations. It uses a relational database and a modular structure that give the platform versatility. An archetypal experiment using gaussian mixture models with expectation maximization and maximum a posteriori is proposed and carried out. The results are within the range of other platforms' performance. Future work will be oriented towards support vector machines and combinations of support vector machines with gaussian mixture models.

KEY WORDS
Speaker Verification, Platform Architecture, Database, Data Selection, SQL, GMM, VAD

1. Introduction

Speaker verification is the process of confirming the validity of a user's claimed identity, using information extracted from the user's voice.

Several methods are available for this purpose, some of them being better under certain circumstances. A *speaker verification platform* provides the tools to compare the outcome of different methods, or combination of methods, in order to minimize the error rate.

Our purpose is to establish a new speaker verification research laboratory focused on applications in the Spanish language. Due to the fact that speaker verification platforms are not publicly available, one of the tasks towards this goal is the development of a new platform.

A major concern with the design of speaker verification platforms is the preparation of experiments, as there is an extensive set of module combinations and recordings from which to select an appropriate subset. This makes preparing experiments a tedious and time-consuming task.

As a consequence, our approach to the design of the platform was focused on providing a flexible and easy to use environment. Notwithstanding, it is important not to trade performance for flexibility, so that the time saved in experiment planning is not wasted on data processing.

These concerns led to a highly modular implementation, where each module consists of one or several of the following components: database management, signal conditioning (voice activity detection), feature extraction (MFCC, LPC), feature modeling (GMM and GMM+UBM), normalization (Z-Norm) and score evaluation (likelihood ratio).

These components were chosen so as to be able to run a baseline experiment. Such an experiment consists in generating client models (and, in some cases, a world model) and performing tests that confront clients to impostors.

In this paper we shall discuss how modular flexibility and the use of a relational database can save time and other resources when preparing experiments and when expanding the platform's functionalities and algorithms.

2. Platform

2.1. Architecture

The implementation was developed in C++, due to the language's high processing performance, power of abstraction, availability of libraries, and easy interfacing with other programming languages (especially C).

Following the UNIX methodology, modules were devised as simple stand-alone command-line applications. Communication among models was implemented using a folder hierarchy with standardized configuration files.

A common problem while writing applications in C language is that programmers have to deal with language issues, and may lose the perspective on the original problem. To overcome this, a high abstraction layer was written, providing the basic structures to be connected in order to develop new modules. This layer consists of a set of C++ classes. Of these, the two most important are f*eatures* and *models.* They provide dynamic typing to allow for model and feature changes at run-time. This gives each module the ability to behave differently depending on an input configuration file.

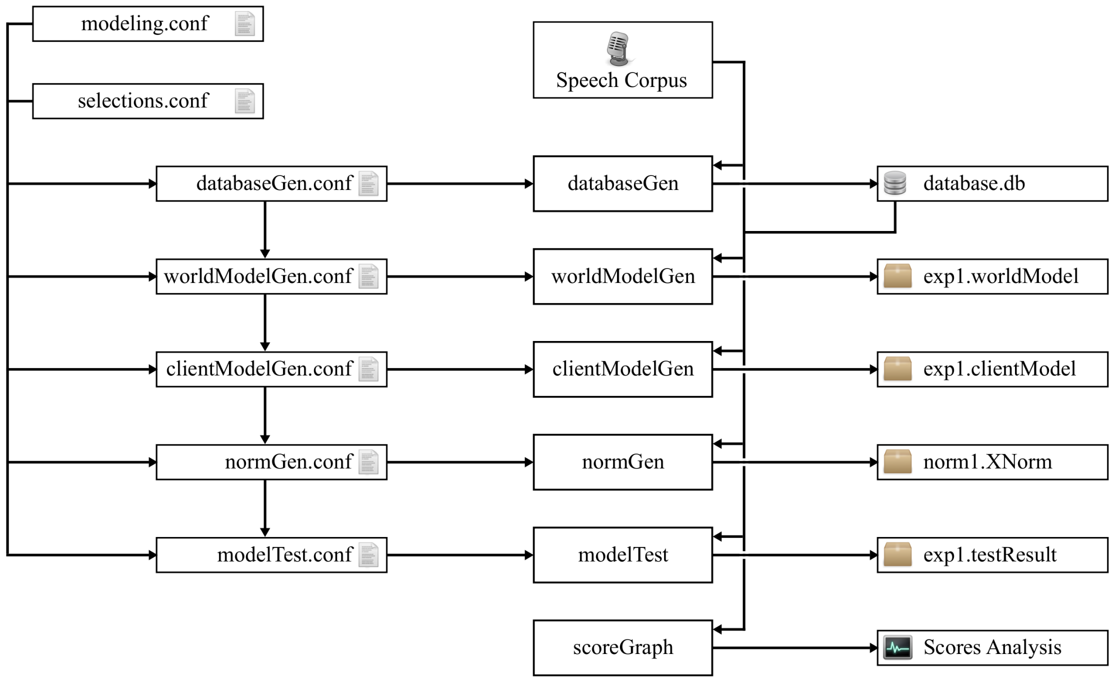


Figure 1: Platform architecture

Classes define the interface between algorithms and modules. This interface is defined in such a way that it is as generic as possible, and all algorithms must follow the structure that it proposes.

The *feature* class allows the representation of generic features, offering methods to extract, store and recall information in files.

Likewise, the *model* class helps to represent generic models, giving enough functionality to train, test, save and recover models.

Regarding feature extraction, the implementation of *mel-frequency cepstral coefficients* (MFCC) and *linear prediction filter coefficients* (LPC) was realized using the *HTK* library [1]. The *Torch3* [2] library was used for implementing *gaussian mixture models* (GMM) [3] and *support vector machines* (SVM) [4]. Both libraries are working behind a wrapper that binds them to the proposed interface.

Matrix and vector algebra was performed using the C++ *Boost* library [5], due to its high efficiency [6] and level of abstraction.

2.2. Database

In order to allow the selection of recordings from a speech corpus, a relational database was created, including all relevant data for each speaker (e.g. sex, age) and recording (e.g. environment, handset and duration).

The database was implemented using *SQLite* [7], an embedded relational database management system that uses the SQL language.

We counted with the *SpeechDat II* corpus, made available to us through a collaboration agreement with UPM (Universidad Politécnica de Madrid). This is an annotated corpus of recordings, where annotations are stored in SAM label file format [8]. In order to populate our relational database (which not only provides independence from the corpus' internal structure but also presents an expanded range of options for data selection) we produced the module called *databaseGen*, that imports all relevant data into the SQLite environment.

SQLite has a remarkable flexibility in the formulation of selection statements. It is possible to define limits with logical expressions in such a way that selections can be specified by duration and classified according to any desired characteristic. This gives, for instance, the ability to separate client model training and test data.

Furthermore, SQLite has a sub-query feature that greatly enhances data filtering. For example, the amount of results for a selection of recordings can be limited independently for each speaker or group of speakers. Moreover, results can be filtered according to rules specified by other results.

2.3. Voice activity detection

The SpeechDat II corpus is oriented towards speech recognition applications, implying that recordings may have periods of silence. As these silences are nothing but noise for our models (because silence is not speaker-dependent), we had the need to produce a *voice activity detector* (VAD) in order to eliminate them.

When choosing an appropriate VAD algorithm, we focused, on the one hand, on finding one that had been extensively tried and tested. On the other hand, the algorithm had to have a low rate of false negatives, that is, it should privilege conserving speech rather than being overzealous.

After a survey of several algorithms, it was found that the voice activity detection of the 3GPP AMR [9] (adaptive multi-rate) speech codec was the most appropriate for our purposes. This algorithm is used to reduce bandwidth requirements in GSM and UMTS communications, and fulfills our requirements of consistency and low zeal.

2.4. Processing modules

A particular format was established for data exchange between modules. We defined a folder hierarchy where the type of contained information is specified by an extension assignment code (for example, a folder named "experiment.worldModel" would contain a world model for a certain experiment).

Each hierarchic entity (or *package*) contains a configuration file. This file contains in itself all information about both previous and current configuration parameters. This self-contained history allows a straightforward administration of the chain of processing events.

A module's configuration parameters are defined through command-line arguments in a key-value fashion. In addition, parameter configuration files can be created and included to a command's arguments. When such a file is entered through the command-line, it is appended to the module's configuration and, therefore, to the affected packages' configuration files. This allows for a quick replication of parameters that extends the reach of a parameter definition or redefinition.

Most modules were structured in a similar fashion, following the aforementioned input/output scheme with its corresponding package hierarchy. This is illustrated in Figure 1.

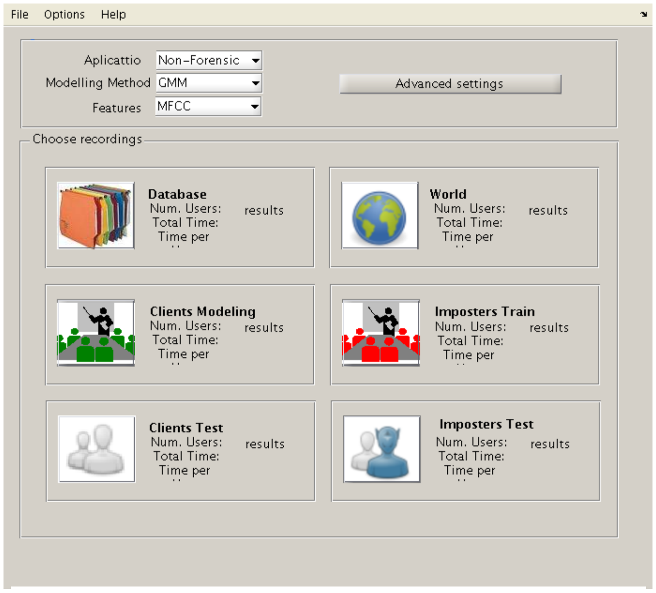
As we mentioned, our immediate goal was to provide the capability to run a baseline experiment. This led to the development of the following modules:

worldModelGen generates a world model for those modeling algorithms that require the representation of a generic speaker. This module receives a selection of recordings from the database as input, and outputs a .worldModel package containing the used configuration and a corresponding model file.

clientModelGen generates a set of client models, either based on a world model or from scratch. The output .clientsModel package contains a common configuration file and separate model files for each client.

normGen generates the normalization information for each model in the .clientsModel folder. Both the output folder's contents and its extension depend on the type of normalization.

modelTest tests every model against recordings from the same person and from a set of impostors. The results are



Aplicattio: Non-Forensic
Modelling Method: GMM
Features: MFCC

Figure 2: User interface module

stored in a folder with the extension .testScore, with two files for each model, one containing the scores (in CSV format) coming from the same person, and the other containing the scores from the impostors.

2.5. Score analysis module

The analysis of experimental results consists mainly in: obtaining probability distributions for client and impostor scores, determining *Tippet* [10] curves, *DET* [11] curves on a linear and logarithmic scale, and finally the *equal-error rate* (EER) [12].

This task is carried out by a module called *scoreGraph*.

2.6. User interface module

The platform's front-end consists of command-line executables and an integrated GUI module (Figure 2). On the one hand, this means that the platform can be used to the full extent of its potential because of the command-line's versatility. On the other hand, the user can count with a simple environment that provides independence from the internal structure of the platform.

Using the GUI module, a researcher can easily generate experiments, consult the database and perform data selection without the need for a deep understanding of the SQL language or the intricacies of the platform.

The module also ensures consistency in the database selection. For instance, it validates that no recording is used more than once.

3. Archetypal experiment

With the purpose of exemplifying the platform's functionality and, at the same time, exhibiting its results, we propose an experiment that typifies the speaker verification process.

The proposed experiment uses several subsets of the

```
include = database.conf
include = modeling.conf

# Output
worldModel = world

# Selection
include = selections.conf
```

Figure 3: modelWorld.conf for archetypal experiment

```
# Features
features = mfcc

# Extractor
extractor = htk

# Model
model = gmm

# Trainer
trainer = torch
torch.train.method = MAP
torch.train.EM.numberOfMixtures = 512
torch.train.maxEMIterations = 100
torch.train.maxKMeansIterations = 100
```

Figure 4: modeling.conf for archetypal experiment

SpeechDat II corpus. 40 recordings from each of 100 speakers (on average, 160 seconds per speaker) were used to generate the world model. For the client-impostor testing, a body of 50 clients and 50 impostors was used. The client models were trained with 10 recordings (on average, 40 seconds per speaker) from each client speaker. Testing was performed with 20 recordings (on average, 4 seconds per recording), from both clients and impostors.

The models were constructed using 39 parameters. Instantaneous energy and 12 MFCC coefficients were used to generate 13 delta coefficients. Likewise these were used to construct 13 delta-delta coefficients.

The world model was built using GMM with *expectation maximization* (EM). A total of 512 Gaussian functions was produced.

The clients were modeled through GMM with a *maximum-a-posteriori* (MAP) probability algorithm. The start point for this modeling was the world model.

The tests were normalized against the world model.

3.1. Platform configuration

Figures 3 and 4 shows code from the configuration files used for the aforementioned experiment.

The flexibility of the whole process is seen in the “include” statements. The code demonstrates how database selection and modeling parameters can be decoupled from a particular module.

A remarkable aspect is how configurations are inherited and shared between modules.

If, for instance, a user wanted to implement LPC coefficient extraction, all that should be done is changing the “features” statement in the configuration file. This would automatically affect the entire hierarchy. As a result, the generation of new experiments with variations in the setup becomes an easy task.

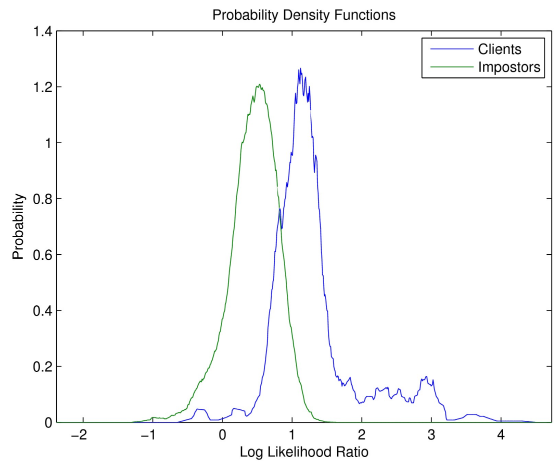


Figure 5: Probability density function

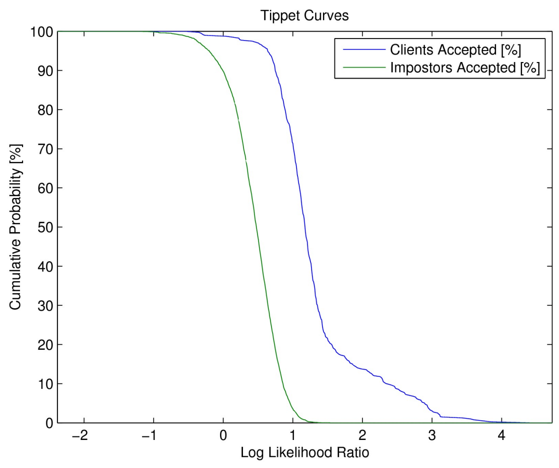


Figure 6: Tippet curve

3.2. Results

The results obtained from the experiment can be seen in Figures 5 - 8. At the current stage, the most representative parameter to evaluate a platform's performance is the equal error rate. As Figure 7 shows, this particular experiment resulted in an EER of 13.5%.

These results were compared to NIST SRE08 [13]. It was determined that the “Telephone Speech In Training and Test” SHORT2-10SEC test was the most suitable for comparison to our archetype experiment, due to a similarity of conditions. Tested platforms from NIST exhibit EER values between 9% and 27%.

However, the conditions for our experiment are more restrictive, since training and testing time for our models is shorter. Considering this, an EER of 13.5% is a more than acceptable value.

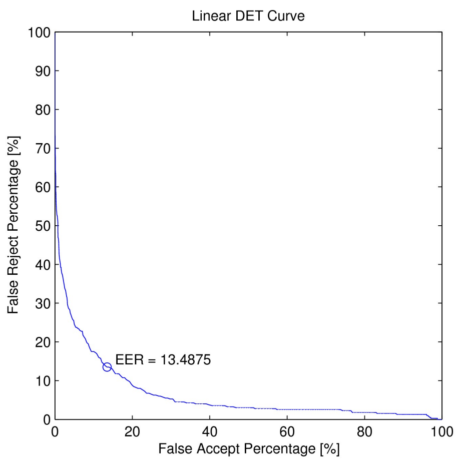


Figure 7: Linear DET curve

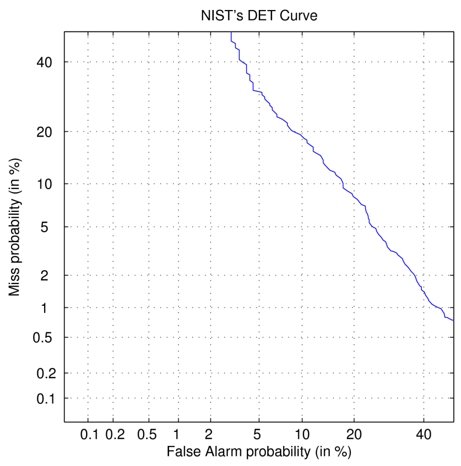


Figure 8: Logarithmic DET curve

4. Conclusions

To conclude, a highly modular and flexible speaker verification platform was produced. It was implemented in C++, presenting efficient processing capabilities and a high level of abstraction. Starting from two base classes (*features* and *models*), a module structure was built allowing a versatile manipulation of a relational database of speakers and speech recordings. These modules carry out the process of speaker verification using a hierarchic package structure as input/output scheme.

To evaluate the platform's ability to produce meaningful results, an archetypal experiment was proposed and executed. The obtained results fall between the range seen in NIST testings of 2008. Still, they are not top-of-the-line, yet the platform's versatility ensures that new techniques can easily be designed, implemented and applied in order to improve performance. Current work is leading towards implementing SVM modeling, SVM with GMM and GLDS, MLLR extraction and APE [14] curve generation for auto-calibration.

Notwithstanding, these results are not enough to perform a solid comparison with other platforms: future work should focus on testing with a standardized corpus and producing a NIST-compatible output.

References

[1] S. Young, D. Ollason, V. Valtchev, and P. Woodland, *The HTK Book (for HTK Version 3.4)* (Cambridge, UK: Cambridge University Engineering Department, 2006).

[2] R. Collobert, S. Bengio, and J. Mariéthoz, *Torch: a modular machine learning software library. Technical Report IDIAP-RR 02-46* (Maritgny, CH: IDIAP, 2002).

[3] D.A. Reynolds and R.C. Rose, Robust text-independent speaker identification using Gaussian mixture speaker models, *IEEE Transactions on Speech and Audio Processing*, 3(1), 1995, 72-83.

[4] W. M. Campbell, J. P. Campbell, D. A. Reynolds, E. Singer, P. A. Torres-Carrasquillo, Support vector machines for speaker and language recognition, *Computer Speech and Language*, 20(2-3), 2006, 210-229.

[5] *uBlas Boost Library*, (http://www.boost.org/doc/libs/1_44_0/libs/numeric/ublas/doc/index.htm).

[6] *uBlas Benchmark Results*, (http://www.boost.org/doc/libs/1_44_0/libs/numeric/ublas/doc/overview.htm#7BenchmarkResults).

[7] *SQLite*, (http://www.sqlite.org).

[8] A. Fourcin et al. *ESPRIT project 2589 (SAM) multi-lingual speech input/output assessment, methodology and standardization. Technical Report SAM-UCL-G004,* (Sam Consortium, 1992).

[9] 3GPP, *TS 26.071 - AMR speech codec: General Description, Version 7.0.0*, (3GPP, 2007).

[10] C. F. Tippet, The evidential value of the comparison of paint flakes from sources other than vehicles, *Journal of the Forensic Science Society*, 8(2-3), 1968, 61-65.

[11] A. Martin, G. Doddington, T. Kamm, M. Ordowski, M. Przybocki, The DET curve in assessment of detection task performance. *Proc. Eurospeech '97*, Rhodes, GR, 1997, Vol. 4 1899-1904.

[12] Jyh-Min Cheng, Hsiao-Chuan Wang, A method of estimating the equal error rate for automatic speaker verification, *2004 International Symposium on Chinese Spoken Language Processing*, Hong Kong, CN, 2004, 285-288.

[13] National Institute of Standards and Technology, *The 2008 NIST speaker recognition evaluation results* (http://www.itl.nist.gov/iad/mig/tests/sre/2008/official_results/index.html, 2008).

[14] Niko Brümmer A, Johan Du Preez, Corresponding author Application-Independent Evaluation of Speaker Detection, *Computer Speech & Language*, 20(2-3), 2006, 230-275.