Implementation aspects of speaker recognition using Python language and Raspberry Pi platform

Radoslaw Weychan, Tomasz Marciniak, Adam Dabrowski
Poznan University of Technology
Faculty of Computing Science
Chair of Control and Systems Engineering
Division of Signal Processing and Electronic Systems
E-mail: radoslaw.weychan@put.poznan.pl

***Abstract*—The paper presents implementation aspects of real time speaker recognition from Internet radio broadcasts. The proposed solution is based on the free and open source Python development tools. The prepared software was tested with the Windows environment and then adapted to the Unix operating system running on the Raspberry Pi platform. We show what libraries are most convenient in order to implement individual blocks of the speaker recognition algorithm. In the paper we also indicate parameters, for which the algorithm exhibits the greatest efficiency. The prepared software is available on the Github file repository.**

***Keywords*—Speaker recognition, GMM, Internet radio, Python, Raspberry Pi**

I. Introduction

This paper aims to show an efficient implementation of the Internet radio receiver with fast speaker identification functionality. Low budget (at a price below $50) minicomputers (such as BeagleBoard, Banana Pi, Raspberry Pi, or Creator CI20) allow to build embedded devices, and thus easy and efficient implementation of digital signal processing algorithms. These platforms support Internet communication via wired or wireless interfaces and may replace expensive specialized DSP platforms [1].

Identification of speakers may be an additional functionality of modern radio receivers that on the LCD display can even offer relevant information about the speakers regardless of sending such an information by the broadcaster. Currently, according to the authors' knowledge, no radio station transmits such an information. Identification functionality should be implemented directly in the embedded system, because it was assumed that the information about speaker may change on the radio display e.g. every second. For this reason, a communication with remote servers (like in case of popular speech recognition Dragon Dictation or FlexT9) would be ineffective because of communication delays.

Basing on the previous authors' studies the speaker recognition functionality has been implemented using standard techniques: MFCC (mel frequency cepstral coefficients) [2] and GMM (Gaussian mixture model) [3], [4] but adapted to the problem of identifying speakers during the talks of two or more people. For such a conversation short utterances with time duration, often less than one second should be analyzed. Therefore, as already mentioned above, it is assumed that the information displayed on the LCD screen about the current speaker may change every second.

The research presented in the papers [5], [6] indicates that the lossy encoding of speech signal contributes to some effectiveness reduction of speaker recognition. Taking advantages of the fact that speaker models obtained with relevant encoders give better results than in other cases[7], [8], it is possible to reduce the EER (equal error rate) by approximately 5-10 % and thus to improve identification efficiency.

The currently presented system preserves functionalities of our experimental Matlab-based system introduced during the previous SPA conference [9]. Now, the proposed solution is realized with the Raspberry Pi platform programmed with the use of the Python language.

II. System description

An idea of the system is to process the input audio signal and display the information about the current speaker in real time, i.e., the same time when the input audio speech signal is being displayed.

The signal processing is done with the use of the open source platform, used in the embedded system. The general idea of the system is presented in Fig. 1.

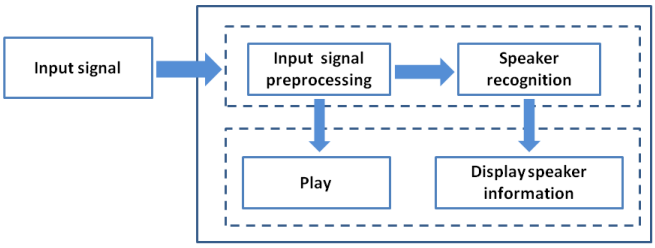

Fig. 1. General idea of the real-time speaker recognition system

One of such open source hardware platforms is the Raspberry Pi [10]. The second edition of the Raspberry Pi board is equipped with four core ARM Cortex-A7 900 MHz processor and 1 GB RAM memory. Four USB ports, LAN port, HDMI and audio outputs make it useful to IoT (Internet of things) applications. This board runs under the Linux Debian operating system, making programming in C++ and Python available and easy. It supports hardware floating point arithmetic operations,

which are very important in the case of Python programming language. There are a lot of libraries and packages for algebra and signal processing available for Python [11]. It makes prototyping and developing of embedded systems faster and easier. Thus the Python programming language was chosen to develop the presented system. The latest version is actually 3.4, however, some scientific libraries require Python 2.7 [12].

Fig. 2 presents the experimental hardware of the system based on Raspberry Pi 2 and LCD screen.

Fig. 2. Experimental hardware of speaker recognition from the Internet radio

The data processing in the presented system can be divided into the following tasks:

1) periodic collection, decoding and playing the Internet radio data
2) feature extraction from the decoded data
3) calculation of features log-likelihood under each speaker model from the database
4) selection of the best match and displaying the result.

The model-based design of the software implementation in both Windows and Unix environments is presented in Fig. 3.

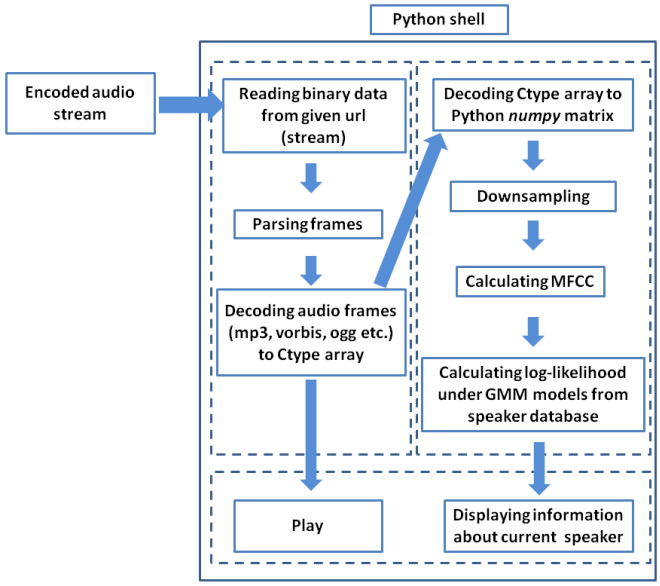

Fig. 3. The detailed automatic speaker recognition system schema

The specific Python 2.7 libraries perform the following tasks:

1) periodic collection of Internet radio data realized by *urllib* standard library. The method *read()* is executed on instance of a library object (the URL address in this case) — in every given moment, a previously set number of 8-bit values is collected from the stream; this number was set to provide both real time play and the speaker recognition task
2) signal framing and decoding, realized with the *pymedia* library [13] — it allows to decompress the stream of data encoded with mp3, ogg, ogg vorbis or wma audio encoders / containers. Despite the fact that *pymedia* project has not evolved since 2006, there is no alternative solution for decoding the streams; most of popular libraries and packages, like e.g. *gstreamer*, operate only on files, thus making them feasible with the stream processing is a toilsome task and requires much interference in the source code — therefore the *pymedia* library was chosen (it requires a precompiled source of the sound encoders to operate properly)
3) displaying audio signal is also done with the *pymedia* library in the Windows environment. In the case of Unix environment on the Raspberry Pi platform, this library is not compatible to work with the Linux ALSA audio driver. Consequently, there is a need to choose another library. The obvious solution is *pygame* [14] in this case, preinstalled on the system. It allows basic operations on vectors of data interpreted as audio signals (play, stop, queue, channels mixing, etc.):
4) conversion of the decoded data to the *ndarray* data format is operated with the *numpy* [15] library for fast operation on matrices, is done with one of the Python native conversion possibilities — it allows to convert instance of any object into iterable string of characters; after that, a set of 8-bit data can be parsed and converted to numeric values with the use of the

binascii standard library
5) decimation is realized with the use of basic *numpy* operations like *mean* and *reshape*
6) calculation of MFCC's is performed with the use of *numpy.linalg* library, which includes linear algebra methods like FFT, DCT, and dot product of matrices
7) data modeling (in the training part) and also calculation of log-likelihood score under the given GMM (in the testing part) is performed with the use of machine learning library *sklearn.mixture* [16]. The library includes methods related to GMM, nearest neighbour (k-NN), artificial neural networks (ANN) and Beyes classifiers.

Beside the real-time recognition system, speaker model generation tool was also developed. Two options are available - to generate the model from data stream or from file saved on hard disc. The principles of work is basically the same in comparison to Fig. 3. The only difference takes place in the last part, the previously calculated MFCC values are modeled with the use of the GMM EM (Gaussian mixture model, expectation maximization) algorithm. Obtained μ, σ and w coefficients, stored in vectors of length 32 (the number is equal to the number of Gaussians), are saved as the dictionary data type in *.mat* file with the use of *scipy* [17] library. It allows to use the obtained values also in the Matlab environment. The dictionary of speaker models is loaded at the beginning of operation of the real-time speaker recognition system.

The proposed system is available on the Github repository [18]. It contains the following files:

1) Internet_radio_speaker_RPI.py — Internet radio speaker recognition software (Python 2.7) designed to run on Raspberry Pi
2) Internet_radio_speaker_windows.py – Internet radio speaker recognition software (Python 2.7) designed to run on under Windows environment
3) MFCC.py — a module for feature extraction developed by Maigo Yun Wang, based on Matlab VOICEBOX [19]. In the described project signal parameters were changed and energy normalization of mel frequency filters was performed
4) Offilne_speaker_recog_TIMIT.py — software used for experiments with the use of TIMIT speaker database [20]. It contains also illustrative solutions for Dirichlet process Gaussian mixture model (DPGMM and VBGMM) [21]
5) Record_and_model.py — software used to record stream and generate a model. Wav-like converted files are stored in the *People* directory
6) *.mat files — contains the obtained, illustrative speaker databases.

III. System performance

The experiments were performed in two stages:

1) examination of efficiency of the algorithm and minimization of computations; the experiments were performed on personal computer (PC) and TIMIT speakers database [20], which contains recordings of 630 individuals, each one spoke 10 short sequences, the sampling rate of the recordings was set to 16 kSps. 9 recordings from each individual were taken to generate the model in training part, and 1 in the testing part (recognition).
2) for a set of system configuration parameters (number of FFT points, sampling rate, data type - described in details below) used in examination of efficiency part, time parameters were also examined, running the system directly on Raspberry Pi platform.

The input system parameters are as follow:

1) data type set to float64, which is a native format for Python language
2) in the case of offline performance examination, sampling rate of input files was 16 kSps. In the case of real time system, sampling rate depends on Internet radio - in this particular case it was 44.1 kSps. The decoded signal was downsampled 4 times to 11.025 kSps to improve calculation speed of MFCC
3) 2048 - point FFT used in MFCC calculation part
4) non energy-normalized triangular mel filterbank (each bandpass filter reaches peak in 1, but the energy of each subsequent band becomes more and more than 1). This kind of triangular filters are used in packages like Matlab VOICEBOX
5) number of MFCC to calculate was set to 13, wherein the first one is omitted
6) data are modeled with the use of 32 Gaussians.

In the case of the real-time implementation on the Raspberry Pi platform, 15000 bytes are collected from the data stream on every loop. It is equivalent 0.9 to 1.6 s of audio signal, depending on compression rate. This is the interval time for recognizing the speaker. The signals of shorter length cannot be used because of necessity to generate statistical model, which requires a suitable number of samples.

Efficiency of the presented system examined on a personal computer was 92.86 %. Results of the time measurement of the particular parts of the software running on the Raspberry Pi is as follows:

1) decoding and conversion to *ndarray* — 35 - 48 ms
2) conversion to play by pygame queue module — 10 - 15 ms
3) decimation — about 3 - 5 ms
4) calculation of MFCC — 200 - 270 ms
5) Log-likelihood — 4 - 6 ms for every speaker in the database (32 Gaussians).
6) displaying the speaker information — 20 ms

It can be noticed, that the most time consuming part is the calculation of MFCC's because of a large number of MAC (multiply and accumulate) operations while performing FFT and filtering. This is the task to be examined to find parameters, which reduce time consumption and do not decrease system accuracy. In the case of software development, the most important part is to handle low-level data with Python. The data returned by the decoder are not iterable, what makes impossible to merge 8-bit values into 16-bit audio samples, and then extract the left channel as mono signal. Thus a series of time consuming operations of typecastings, parsing and

decoding of single values are necessary. Direct implementation of above mentioned set of tasks take up to 800 ms, so it is important to use well optimized C/C++ operations performed by *numpy* library. The conversion schema is presented in Fig 4.

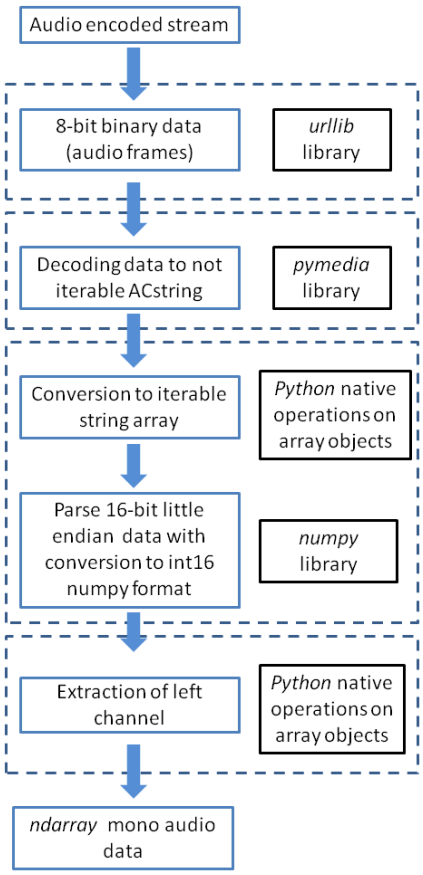

Fig. 4. The detailed flow of low-level data conversion

The on-board CPU consumption of the above process is 3 up to 6 %. The main program can run in the background, while the processor handle other tasks. The system consumes 45 MB of RAM memory of 1 GB available. Table I presents the detailed information of software memory allocation. An important fact is to handle the Python start point at 0x8000 address which is where the ARM processor starts execution of machine code.

TABLE I. MEMORY MAP OF THE PRESENTED SYSTEM

Adress	Data size	Data type
0x8000	2630 KB	Python 2.7
0x82b000	23680 KB	Heap
0x72200000	40000 KB	Stack, Linux & Python libraries (*.so)

The executable part of the software takes about 6 MB, what can be seen with the use of *memory profiler* tool for Python. The main program starts with 38.805 MB of RAM consumption, and while running the peak is equal to 44.305 MB.

The first experiment with the aim of reducing the execution time was to change data resolution from 64-bit floating point to 32-bit. Taking into consideration, that ARM-7 includes hardware support for 64-bit data processing, time parameters were not better after the reduction of resolution. The next reason is that some of calculations inside the Python libraries are performed with the use of 64-bit data, thus additional typecasting to 32-bit resolution takes additional time. Still another important fact is that Python numerical *numpy* library prefers 64-bit data and object generation, so calculation on them can be faster. Finally, the resolution of data in the presented system was set to 64-bit floating point.

Table II presents results of other experiments aimed to increase (or at least not decrease) the system accuracy while reducing the number of the performed calculations. One of the fields was marked red because of high efficiency and significant reduction of computations.

TABLE II. OFFLINE PERFORMANCE

Test number	Sampling rate [kSps]	Normalization of mel filterbank	FFT resolution	Number of Gaussians	Efficiency [%]
1	8	NO	2048	32	77.3
2	8	YES	2048	32	77.62
3	16	NO	2048	32	92.86
4	16	YES	2048	32	93.02
5	16	NO	1024	32	93.17
6	16	YES	1024	32	93.6
7	16	NO	512	32	92.85
8	16	YES	512	32	92.85
9	16	NO	256	32	92.54
10	16	YES	256	32	92.54
11	16	YES	256	64	91.11
12	16	YES	256	16	92.38
13	16	YES	256	8	83.97
14	16	NO	128	32	86.5
15	16	YES	128	32	87.62
16	16	NO	64	32	64.12
17	16	YES	64	32	64.76

It has to be noticed that the use of the energy-normalized mel frequency filterbank increases the system accuracy in general. The normalization is performed at the beginning of the program, before the real-time processing. Thus it does not influence the time of computations.

The next observation concerns the increasing of the sampling rate which improves the system accuracy. While the sampling rate 44.1 kSps is too large to represent speech [3] (most of the Internet radios use this sampling rate or 48 kSps), the offline experiments were done with the use of of 8 kSps and 16 kSps. In the case of real-time application the basic sampling rate 11.025 kSps (downsampled 4 times from 44.1 kSps) was increased to 22.050 kSps in some cases (the input signal was downsampled twice), listed in Table III. It can be predicted that this operation will also significantly increase the system accuracy.

It can be observed that reduction of the FFT resolution even to 256 points do not significantly influences the system accuracy (reduction is equal to 1 % only). Below this value accuracy of the system is decreasing even to about 65 %. Thus the FFT resolution was set to 256 points.

The last experiment was to investigate the influence of the number of Gaussians on the system accuracy. The starting value was 32. Reduction of this number can improve timing parameters of the log-likelihood parameters. The experiments were done with the number of Gaussians set to 64, 16 and 8. Double reduction of the number of Gaussians does not significantly decrease the accuracy in comparison to value 32 (about 0.2 %). Thus parameters of these experiments were red marked as prospectively the best under conditions of time and accuracy parameters combination.

Table III presents measurement of time of specific parts of the real-time application. In the "Time measurement" column, particular numbers (1 - 5) are directly related to listing of first experiment in Section III, where main parts of the software were measured. The time of displaying of the speaker information is not important in this case, thus it was omitted. The chosen system parameters are a subset of listed in Table II. The aim of the experiments was to investigate the influence of the sampling rate and the FFT resolution on the general time consumption.

TABLE III. COMPARISON OF TIME MEASUREMENT UNDER VARIOUS CONDITIONS

Test number	Sampling rate	FFT resolution	Number of Gaussians	Time measurement [ms]	Total max time [ms]
1	11025	1024	32	1) 35-48 ms 2) 10-15 ms 3) 3-5 ms 4) 95-145 ms 5) 4-6 ms	219
2	11025	512	32	1) 35-48 ms 2) 10-15 ms 3) 3-5 ms 4) 60-80 ms 5) 4-6 ms	154
3	11025	256	32	1) 35-48 ms 2) 10-15 ms 3) 3-5 ms 4) 40-60 ms 5) 4-6 ms	134
6	11025	128	32	1) 35-48 ms 2) 10-15 ms 3) 3-5 ms 4) 40-60 ms 5) 4-6 ms	134
7	11025	64	32	1) 35-48 ms 2) 10-15 ms 3) 3-5 ms 4) 40-55 ms 5) 4-6 ms	129
4	22050	256	32	1) 35-48 ms 2) 10-15 ms 3) 4-7 ms 4) 50-70 ms 5) 4-6 ms	146
5	22050	256	16	1) 35-48 ms 2) 10-15 ms 3) 4-7 ms 4) 50-70 ms 5) 2-4 ms	144

In the case of 11.025 kSps sampling rate, time consumption can be decreased by 91 ms down to 930 ms, but the system accuracy will be significantly lower in comparison to the sampling rate 22.050 kSps. Thus more important, from the point of view of the system accuracy, were time measurements for this particular sampling rate value. The number of the FFT points was set arbitrarily to 256 as a consequence of the previous experiments, listed in Table II. For 32 Gaussians, the general time consumption is higher than in the related experiment of 11.025 kSps sampling rate. Reduction of Gaussians number will speed up algorithm for 2 ms (50 % faster log-likelihood calculation). It has to be taken into consideration that the speaker database will contain even hundreds of speakers. As the system accuracy does not decrease for the Gaussian reduction to 16, the time consumption significantly decreases when a large database is being used. The final system parameters are marked in the red box in Table III.

IV. FUTURE WORK

The presented system operates in real time, but there is still place for improvements like handling more external devices or to add preprocessing algorithms. At the moment, only one core of the ARM Cortex-A7 is in use. In addition more experiments can be provided with some extended modeling algorithms like supervector UBM (universal background model) transformed to i-vectors [22]. The low-level source code optimization should also be considered.

V. CONCLUSION

In this paper the results of the time optimization of the real-time speaker recognition system were presented. The obtained parameters prove that the system accuracy can be held on the same level (or even increased) while reducing the number of computations related to the MFCC (8 time less FFT computations) and GMM (twice less computations) algorithms. As a consequence, the sampling rate can be increased to provide more accurate real time speaker recognition (more than 10 %). Taking an advantage of open source hardware platforms and software (like Raspberry Pi and Python used in this project) developing of inexpensive, fast, and publicly available system is feasible. It is one of the most important advantages in comparison to e.g. professional platforms with digital signal processors. Raspberry Pi 2 was used in this case but the described system can be successfully run on previous versions of this hardware.

REFERENCES

[1] T. Marciniak, R. Weychan, A. Stankiewicz, and A. Dabrowski, "Biometric speech signal processing in a system with digital signal processor," *Bulletin of the Polish Academy of Sciences Technical Sciences*, vol. 62, no. 3, pp. 589–594, 2014.

[2] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, no. 4, pp. 357–366, Aug 1980.

[3] H. Beigi, *Fundamentals of speaker recognition*. Springer Science & Business Media, 2011.

[4] P. Lenarczyk and Z. Piotrowski, "Speaker recognition system based on GMM multivariate probability distributions built-in a digital watermarking token," *Przeglad Elektrotechniczny*, vol. 89, no. 2a, pp. 59–63, 2013.

[5] R. Weychan, T. Marciniak, and A. Dabrowski, "Analysis of differences between MFCC after multiple GSM transcodings," *Przeglad Elektrotechniczny*, pp. 24–29, 2012.

[6] R. Weychan, A. Stankiewicz, T. Marciniak, and A. Dabrowski, "Improving of speaker identification from mobile telephone calls," in *Multimedia Communications, Services and Security*, ser. Communications in Computer and Information Science, 2014, vol. 429, pp. 254–264.

[7] T. Marciniak, R. Weychan, A. Dabrowski, and A. Krzykowska, "Speaker recognition based on short Polish sequences," *IEEE SPA: Signal Processing Algorithms, Architectures, Arrangements, and Applications Conference Proceedings*, pp. 95–98, 2010.

[8] A. Dabrowski, S. Drgas, and T. Marciniak, "Detection of GSM speech coding for telephone call classification and automatic speaker recognition," *ICSES*, pp. 415–418, 2008.

[9] R. Weychan, T. Marciniak, A. Stankiewicz, and A. Dabrowski, "Real time speaker recognition from internet radio," in *Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), 2014*, Sept 2014, pp. 128–132.

[10] Raspberry Pi platform detailed overview. [Online]. Available: http://elinux.org/RPi_Hardware

[11] Numeric and scientific python packages. [Online]. Available: https://wiki.python.org/moin/NumericAndScientific

[12] P. S. Foundation. Python language reference, version 2.7. [Online]. Available: http://www.python.org

[13] Pymedia - python module for multimedia files and streams processing. [Online]. Available: http://pymedia.org/features.html

[14] Pygame project website. [Online]. Available: https://www.pygame.org/wiki/about

[15] S. Van der Walt, S. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficientnumerical computation," *Computing in Science and Engineering*, vol. 11, pp. 22–30, 2011.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–, [Online; accessed 2015-05-29]. [Online]. Available: http://www.scipy.org/

[18] Github source code repository. [Online]. Available: https://github.com/audiodsp/Internet_radio_speaker_recognition

[19] M. Brookes, "VOICEBOX: Speech Processing Toolbox for MATLAB," 2005.

[20] J. S. Garofolo, L. D. Consortium *et al.*, *TIMIT: acoustic-phonetic continuous speech corpus*. Linguistic Data Consortium, 1993.

[21] D. M. Blei and M. I. Jordan, "Variational inference for dirichlet process mixtures," *Bayesian Analysis*, vol. 1, pp. 121–144, 2005.

[22] C. S. Greenberg, D. Bansé, G. R. Doddington, D. Garcia-Romero, J. J. Godfrey, T. Kinnunen, A. F. Martin, A. McCree, M. Przybocki, and D. A. Reynolds, "The nist 2014 speaker recognition i-vector machine learning challenge," in *Odyssey: The Speaker and Language Recognition Workshop*, 2014.

This work was partly supported with the DS 2015 means and partly with the project "Scholarship support for Ph.D. students specializing in major strategic development for Wielkopolska", Sub-measure 8.2.2 Human Capital Operational Programme, co-financed by the European Union under the European Social Fund.