Fast Labeling and Transcription with the Speechalyzer Toolkit

Felix Burkhardt

Deutsche Telekom Laboratories, Berlin, Germany
Felix.Burkhardt@telekom.de

Abstract

We describe a software tool named "Speechalyzer" which is optimized to process large speech data sets with respect to transcription, labeling and annotation. It is implemented as a client server based framework in Java and interfaces software for speech recognition, synthesis, speech classification and quality evaluation. The application is mainly the processing of training data for speech recognition and classification models and performing benchmarking tests on speech to text, text to speech and speech categorization software systems.

Keywords: speech, tools, transcription.

1. Introduction

This software tool was developed at our laboratory originally to annotate large quantities of speech data with emotional labels, see (Burkhardt et al., 2009a) for a description of these projects. Nowadays, the main application is lexical transcription of speech data to train new data models for Automatic Speech Recognition (ASR) and Speech-To-Text (STT) systems.

While ASR is used by automatic dialog systems to interpret the user's utterances and is usually domain-dependent, STT can be used to transcribe arbitrary voice recordings like emails, short messages or radio recordings irrespective of domain.

In contrast to existing audio analysis tools like Wavesurfer (Sjölander and Beskow, 2000), Praat (Boersma, 2001) or similar, our tool is optimized for very fast manual processing of large audio data sets. Several thousands of audio files have been manually labeled with this tool in near real time. It is comparable to platforms such as WebTranscribe by (Draxler, 2005), but was originally focused as a training tool for anger recognition applications. Additionally it is used as an integrated platform for general speech processing applications resulting from our work.

External tools are interfaced as libraries, native libraries, web-services or by calling executable binaries on a file system basis. The Speechalyzer can be used by command line or a client graphical user interface (GUI). It will be released as open source software.

This article is organized in the following sections. Section 2. introduces the main functionality and handling of audio files that is common to all modules. Section 3. discusses the way orthographic transcription and annotation can be done with the Speechalyzer. After this, section 4. describes how the labeling process works. Section 5. explains the interfaces to automatic speech recognition, while Section 6. deals with the speech synthesis capabilities. Finally, Section 7. discusses the use of the Speechalyzer as an audio classification framework. We conclude the paper in Section 8..

2. General functionality

The Speechalyzer is a client-server based application written in Java. Figure 1 displays the system architecture.

The general behavior, visibility of certain modules and file paths are stored in a central configuration file for the server. Additionally, each client is configured via applet parameters. The server manages the audio files based on an internal listing, while several clients might access audio information and modify transcripts and annotations.

The server is implemented as a Java application, the client as a Java applet and therefore runs with any browser and operating system supporting the Java plug-in. Currently Java version 1.6 is required.

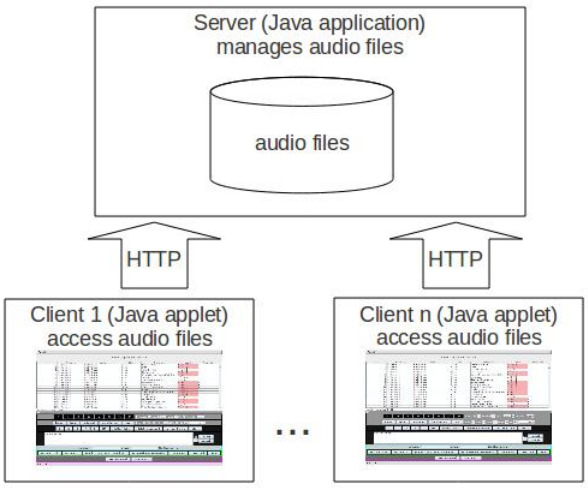

Figure 1: Client server based architecture of the Speechalyzer.

The audio files don't have to be stored physically on the client machine and several annotators can work in parallel on the same database. A user management to avoid access conflicts and perform a rights management is not implemented yet and subject of a future release. Besides the graphical user interface, a command line interface can be used to integrate the tool into automated processes. The list of audio files is loaded by the server at start-up, the files which must be stored on the same machine as the server.

For each audio file, the parent directory's name is automatically used as a "dialog" designator. The term "dialog" is used for a set of audio files that belong to one conversation, e.g. as in a call center interaction.

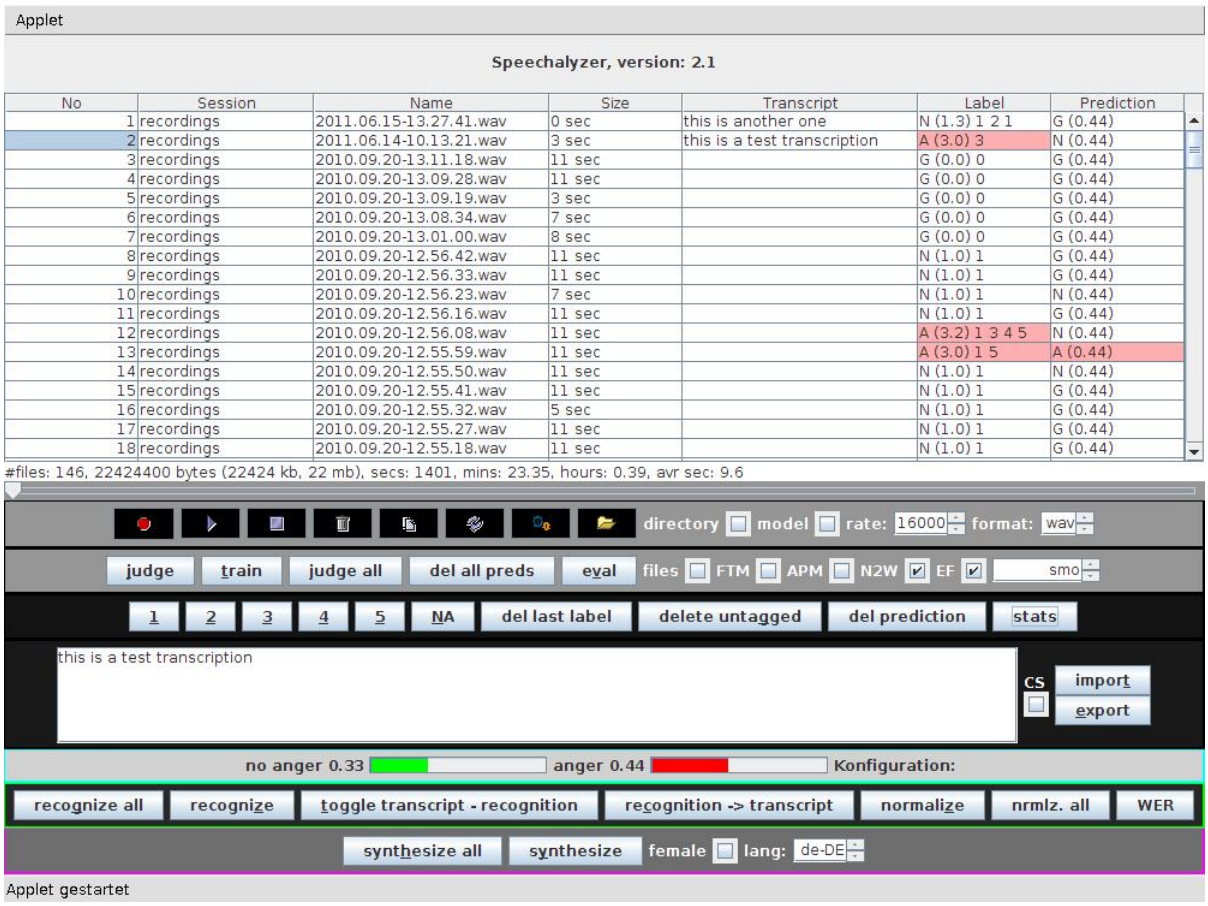

Figure 2: Screen shot of the Speechalyzer client GUI with maximum modules activated

All textual data describing the audio files is stored automatically in accompanying text files which must reside in the same directory and have the same name but different extension as the audio files. These files are in the further text referred to as “data files”.

The GUI (Figure 2) as well as the command line interface offer import and export functionality based on text lists. An example format would be

```
<audio file path> [labels | transcription]
```

which means that you can’t export or import both labels and transcriptions in one step.

Different functionalities, i.e. transcription, analysis, synthesis, recognition or classification, are organized in modules which can be switched on and off in the GUI.

Audio files can also be recorded from a sound device by the client interface. These files can then renamed, recognized, classified or synthesized just like loaded files.

A slider in the client’s GUI shows the playback progress and can also be used to start playback at a specified time in the audio signal. The possibility to view the audio file’s waveform or spectrogram is not foreseen with the Speechalyzer, but one of the Buttons can freely be assigned to call an external program, e.g. a Praat script, with the selected audio file’s path as a command line argument. This does only work if server and client run on the same machine.

3. Transcription and annotation

Transcription as well as annotation can be treated in the same way, as in both cases the audio files have to be linked to an orthographic representation. The tool makes no difference between them, they are both stored in the same manner in the data file and if a user wants to use both, the differentiation has to be handled by the user, e.g. by defining a tag. The tool is optimized for fast transcription by presenting the audio files as rows in a table. In order to avoid mouse movement and unnecessary distraction from the task at hand, automatic playback of the subsequent audio when a transcription has been performed can be activated by the so-called “fast transcription mode”.

Additionally, several functionalities can be used to ease the transcriber’s task:

- Automatic number conversion can be activated and leads to the automatic conversion of numbers into orthographic representations.
- Integrated spell checking can also be optionally activated and leads to a spell check of the last token, i.e. string of characters surrounded by white space. The hunspell (Hunspell,) library is used to perform this step.
- The automatic tagging of words, i.e. the surrounding of tokens by XML-tags (xml, 2008), can be achieved by typing a two-character code followed by the F1 key.

This feature is often useful when transcribing text that shall be used to train speech recognizers based on statistical grammars. Named entities can thus be tagged to generate text variations, e.g. "show me flights from <town/> to <town/>". The codes and tag names are specified in the client configuration.

- In a similar way, the automatic expansion of shortcuts is done via single character codes by typing the code and then the F2 key. Special markings as required in some transcription markups, e.g. "[fil]", denoting a hesitation noise in (Höge et al., 1999), can thus be inserted easily.
- Abbreviations like "eg" or "etc" can also be specified in the client configuration and are automatically replaced after a white space character is inserted.
- Automatic normalization based on regular expression pattern matching rules can be used to normalize the transcriptions as explained further in section 5..
- Lastly, prefixes can be specified and added automatically to the last token by adding a character code and its corresponding prefix to the client configuration, e.g. a "*"-prefix marks a wrongly pronounced word.

The transcription or annotation gets entered in a text input field either directly in the GUI or in a separate window, depending on the client configuration. The "Enter" character finishes the input sequence, this means it can not be part of the transcription.

4. Categorization

Data categorization or labeling, i.e. associating a category from a limited set of classes with the speech data, can be done, like the transcription task, in a fast mode. If this is activated, the program automatically selects and plays the next audio file if the user assigned a label.
The set of buttons that are used to label the audio data, buttons 1-5 and "NA" in the screen shot of Figure 2, can be specified in the client configuration. Several labels or ratings can be assigned per audio file and automatically unified by computation of the mean value. In the server configuration, the set of possible categories are specified as pairs of category descriptors consisting of a numeric value and a character string. Here is an example specification.

```
categories=-1,NA;0,G;1,N;2,A;3
```

The numeric value can be used to unify several labels numerically, the character label can provide for semantic interpretation, e.g. "G" for "garbage", "A" for "anger" and "N" for "non anger". The numeric values are meant as minimum labels, e.g. "1,N;2,NA" means category "N" is assigned for values between $1 >= x < 2$. The list must be in ascending order.
To illustrate what is meant by this, we look at the use case of anger detection as described in (Burkhardt et al., 2009b). The labelers had the choice to assign an anger value between 1 and 5 (1:not angry, 2: not sure, 3: slightly angry, 4: clear anger, 5: clear rage), or mark the turn as "non applicable" (garbage). Garbage turns included a multitude of turns that could not be classified for some reason, e.g. DTMF tones, coughing, baby crying or lorries passing by. We unified the ratings by mapping them to four classes ("not angry", "unsure", "angry" and "garbage") to further process as follows: in order to calculate a mean value for three judgments, we assigned the value 0 to the garbage labels. All turns reaching a mean value below 0.5 were then assigned as "garbage", below 1.5 as "not angry", below 2.5 as "unsure" and all turns above that as "angry".

5. Recognition

Via interface classes, a speech recognition software can be accessed by HTTP protocol to transcribe speech files automatically. This can serve two applications, firstly to be used as a start hypothesis for manual transcription and secondly to benchmark an existing speech recognition system.
All or a selection of audio files can be processed by the speech recognizer and the result gets stored in the data file. Via a toggle button in the client GUI, recognition result as well as transcription can be displayed alternatively. By clicking another button, the recognition result can be used as a transcription hypothesis, as can be seen in Figure 2.
Text normalization can be done automatically be applying a set of regular expression pattern matching rules on the recognition result. The rules are stored in text files on the server. One example for this would be the automatic conversion to lower case of the text, but much more complex transformation is possible by this mechanism, including automatic tagging of named entities and the like.
To evaluate the recognition performance, the NIST word error rate computation script (Pallet, 2003) is interfaced by the Speechalyzer via batch execution. Hypothesis and Reference files are generated automatically from recognition result and transcription and the result is presented to the client GUI in a message window as shown in Figure 3 in the middle window.

6. Synthesis

Text To Speech synthesis (TTS) is integrated to load texts and synthesize audio files via an interface class handling language, voice name, sex and even emotion of the text. Several connectors to commercial and open source synthesizers are implemented. Additionally, the files can be mixed with noise to simulate test conditions like recordings in the open, although we admit that we ignore the implications of the Lombard effect.
This can be used to either perform benchmarks on TTS by judging the results with labels, test ASR modules with the audio files, or use the audio files in dialog system applications. From the client GUI, some or all audio files can be selected for synthesis, of course the audio content will then be replaced. To generate new files, an audio file list with transcriptions can be imported like described in section 2..

7. Classification

Furthermore, the Speechalyzer can also be used to automatically classify speech data via interfaces to several fea-

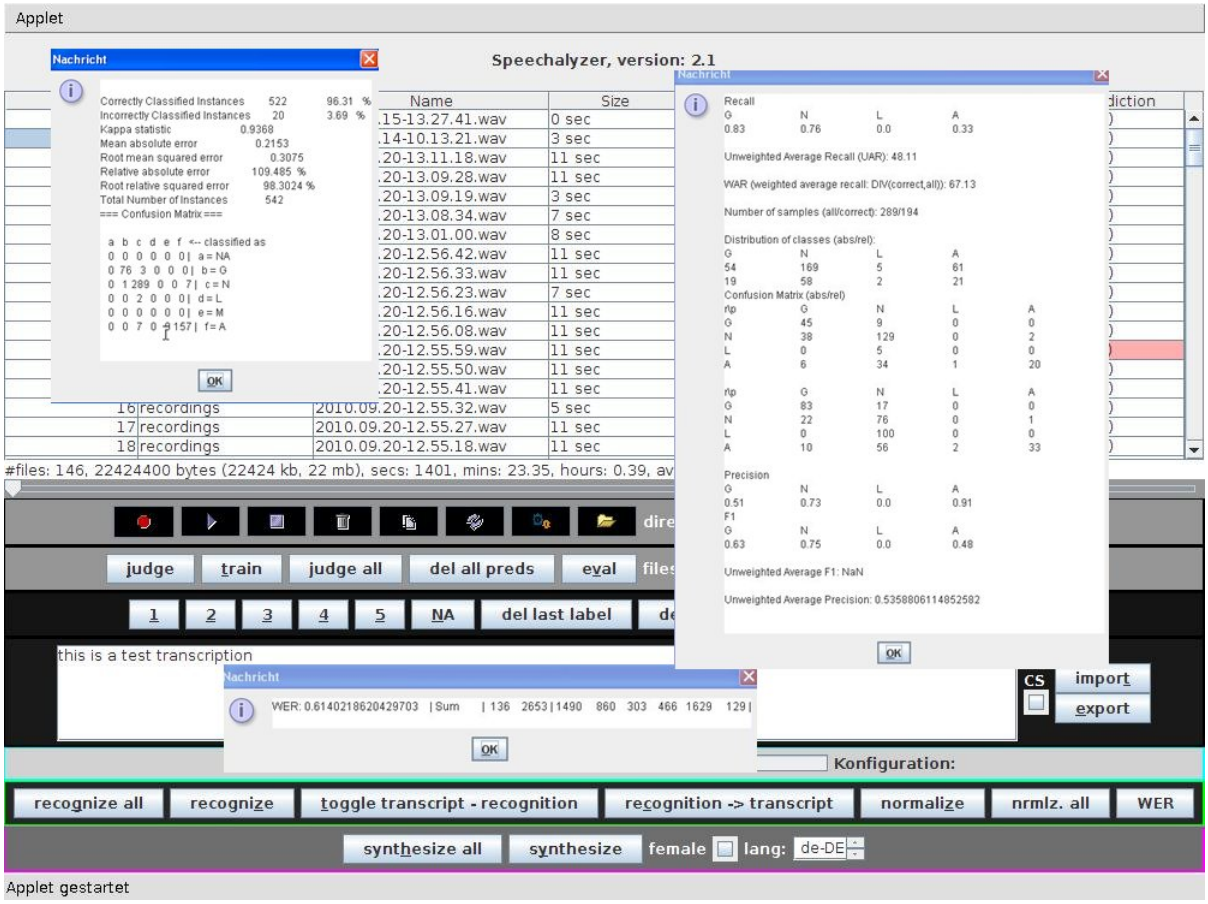

Figure 3: Displaying evaluation output with the Speechalyzer client GUI, a 10 fold cross evaluation, the word error rate and classification statistics.

tures extraction systems and statistical classification systems. We interfaced as feature extractors the Praat software (Boersma, 2001) and the OpenSMILE system (Eyben et al., 2010). As a statistical classification framework, WEKA (Witten and Frank, 2005) is interfaced. When all audio files are assigned to categories as described in section 4., models can directly be trained from the tool.

Several evaluation methods can be applied to the audio data and the models. A ten-fold cross evaluation based on WEKA can be computed on the model as shown in Figure 3 in the left hand message window. Additionally, the audio data can be treated as test data against the current model and a statistical evaluation report generated as shown in Figure 3 in the right hand message window. The Speechalyzer has been used to compute the baseline evaluation results in the 2010 Interspeech Paralinguistic Challenge (Schuller et al., 2010).

8. Summary and outlook

We described a general software tool for speech data implemented in Java. It is optimized for fast labeling and transcription of very large audio file collections, but also gets used as a general framework for all sorts of speech processing like recognition, synthesis and classification.

In future releases we will probably concentrate on the integration of user models, because the Speechalyzer as it is now can hardly be used by several users operating on the same database in parallel. Adapters to store audio and textual data in databases are planned. Furthermore we plan to provide for built in support of the emotional markup language EmotionML (Schröder et al., 2010).

The software will be released as open source which hopefully results in new perspectives for enhancement based on a larger user and developer community.

9. References

Paul Boersma. 2001. Praat, a system for doing phonetics by computer. *Glot International*, 5(9/10):341–345.

F. Burkhardt, K.P. Engelbrecht, M. van Ballegooy, T. Polzehl, and J. Stegmann:. 2009a. Emotion detection in dialog systems - usecases, strategies and challenges. In *Proceedings Affective Computing and Intelligent Interaction (ACII), Amsterdam, The Netherlands*, 9.

Felix Burkhardt, Tim Polzehl, Joachim Stegmann, Florian Metze, and Richard Huber. 2009b. Detecting real life anger. In *Proceedings ICASSP, Taipei; Taiwan*, 4.

Christoph Draxler. 2005. Webtranscribe - an extensible web-based speech annotation framework. In Vaclav Matousek, Pavel Mautner, and Tomas Pavelka, editors, *Text, Speech and Dialogue*, volume 3658 of *Lecture Notes in Computer Science*, pages 747–747. Springer Berlin / Heidelberg.

F. Eyben, M. Wöllmer, and B. Schuller. 2010. openSMILE – the Munich versatile and fast open-source audio fea-

ture extractor. In *Proc. of ACM Multimedia*, pages 1459–1462, Florence, Italy, October. ACM.

Harald Höge, Christoph Draxler, Henk van den Heuvel, Finn Tore Johansen, Eric Sanders, and Herbert S. Tropf. 1999. Speechdat multilingual speech databases for teleservices: Across the finish line. In *Proc. of Eurospeech*.

Hunspell. http://hunspell.sourceforge.net/.

David Pallet. 2003. A look at nist's benchmark asr tests: Past, present, and future. *Proc. Automatic Speech Recognition and Understanding (ASRU)*.

Marc Schröder, Paolo Baggia, Felix Burkhardt, Alessandro Oltramari, Catherine Pelachaud, Christian Peter, and Enrico Zovato. 2010. W3c emotion markup language (emotionml) 1.0. http://www.w3.org/TR/emotionml/.

Bjorn Schuller, Stefan Steidl, Anton Batliner, Felix Burkhardt, Laurence Devillers, Christian Müller, and Shrikanth Narayanan. 2010. The INTERSPEECH 2010 Paralinguistic Challenge. In *Proc. Interspeech*, Makuhari, Japan.

Kre Sjölander and Jonas Beskow. 2000. Wavesurfer - an open source speech tool.

Ian H. Witten and Eibe Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, June.

2008. W3c extensible markup language (xml) 1.0 http://www.w3.org/tr/xml/, 11.