SPEECHALYZER: A SOFTWARE TOOL TO PROCESS SPEECH DATA

Felix Burkhardt

Deutsche Telekom Laboratories
Felix.Bukhardt@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.

1 Introduction

This software tool was developed at our laboratory originally to annotate large quantities of speech data with emotional labels, see [3] 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 [11], Praat [2] 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. 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 or by calling executables on a file system basis. The Speechalizer can be interfaced by command line or a client graphical nuser interface (GUI), which is organized in a single main window. Figure 1 shows a screen shot. Specific modules to perform certain tasks are displayed in areas and can be made visible or hidden in the configuration file. The modules visible in the screen shot are:

- The audio files displayed in table rows with information on length and annotation.
- A module for playback and recording, file opening and selection of audio features like sample rate and file format.
- A module to classify the audio files and train models.
- A module for manual labeling and categorization of audio files.
- A module for manual annotation and transcription of audio files.
- A module to display the result of emotion classification.

- A module to perform speech recognition on the audio files.
- A module for speech synthesis and audio generation.

These modules resp. functionality are described in more detail in the following sections. The Speechalyzer is planned to be released as open source software in the future.

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. The following 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 with a summary and an outlook in section 8.

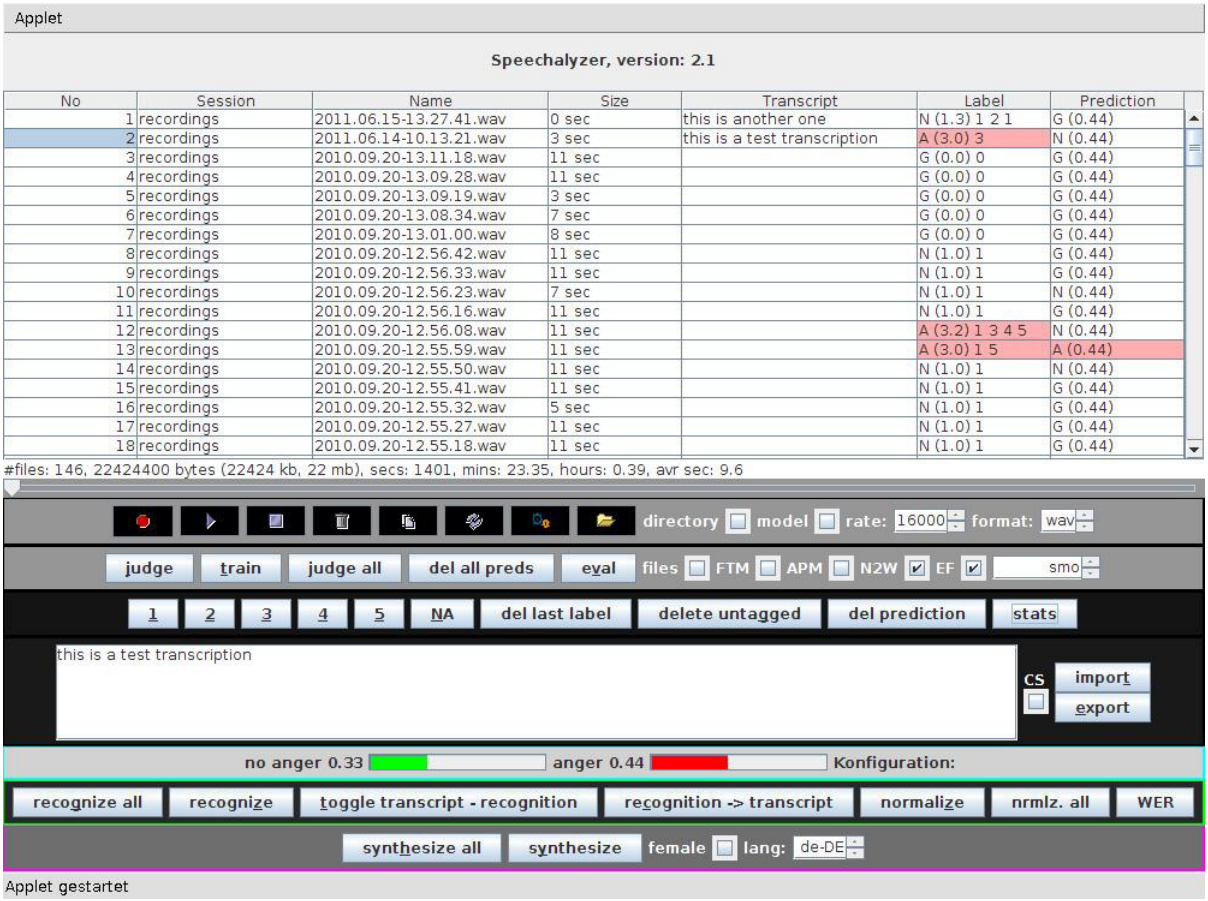

Figure 1 - Screen shot of the Speechalyzer client GUI with maximum modules activated

2 General functionality

The Speechalyzer is a client-server based application written in Java. The general behavior, visibility of certain modules and file pathes 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.

In Figure 2 this architecture is depicted. 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. It has been used on Windows XP, Linux Ubuntu Lucid Lynx and Mac OS Snow leopard, albeit not with all modules active.

This means that 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 as well as calling some extended functionality. The list of audio files is loaded by the server at start-up as a list of audio files which must be stored on the same machine as the server. The audio files might be in different directories, might be organized in a directory hierarchy and the pathes can be absolute as well as relative.

Alternatively, the location of the audio files can be told the server by providing a parent directory. 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.

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 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.

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.

For testing, audio files don't have to be loaded from the file system but can be recorded by the client interface as well. These files can then renamed, recognized, classified or synthesized just like loaded files.

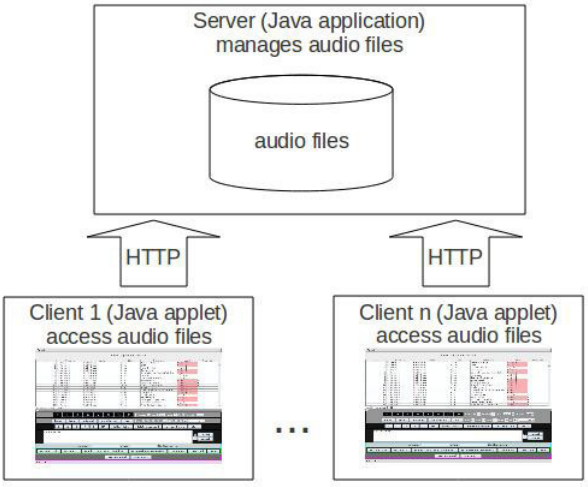

Figure 2 - Client server based architecture of the Speechalyzer.

3 Transcription and Annotation

With “transcription” we mean the manual transformation of the audio signal into an orthographic representation. With “annotation” we mean the association of a written description about some arbitrary feature of the audio files. They 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.

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 functionality 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 [7] library is used to perform this step.
- The automatic tagging of words, i.e. the surrounding of tokens by XML-tags [1], 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 [6], 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 Labeling and Categorization

With “labeling” we mean the association of an audio file with a fixed set of labels. With “categorization” we mean the assigning of an audio file to a fixed set of categories. Both can be treated in the same way, as the audio files have to be assigned to a fixed set of symbols. The tool

makes no difference between them, they are both stored in the same manner in the data file. If a user wants to use both the differentiation has to be handled by the user.

Data labeling or categorization, 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 plays the next audio file if the user assigned a label.

On this behalf, the set of buttons that are used to label the audio data, buttons 1-5 and "NA" in the screen shot of Figure 1, can be specified in the client configuration. Several labels or ratings can be assigned per audio file and automatically unifiied 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 [4].

The labelers had the choice to assign an anger valeu 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 1.

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 [8] is interfaced by the Speechalyzer on the file system basis. 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, i.e. and synthesize audio files via interface classes to several commercial and open source synthesizers. 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 be replaced then. To generate new files, an audio file list with transcriptions can be imported like described in section 2.

From the client GUI, gender and language of the speech synthesis voice can be chosen, of course all this only has effect if the underlying speech synthesis engine supports this features.

7 Classification

Furthermore, the Speechalyzer can also be used to automatically classify speech data via interfaces to several features extraction systems and statistical classification systems. We interfaced as feature extractors the Praat software [2] and the OpenSMILE system [5]. As a statistical classification framework, WEKA [12] 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 [10].

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 [9].

Also the release of the software as open source is planned which hopefully results in new perspectives for enhancement based on a larger user and developer community.

References

[1] *W3C Extensible Markup Language (XML) 1.0 http://www.w3.org/TR/xml/*, 11 2008.

[2] BOERSMA, P.: *Praat, a system for doing phonetics by computer.*. Glot International, 5(9/10):341–345, 2001.

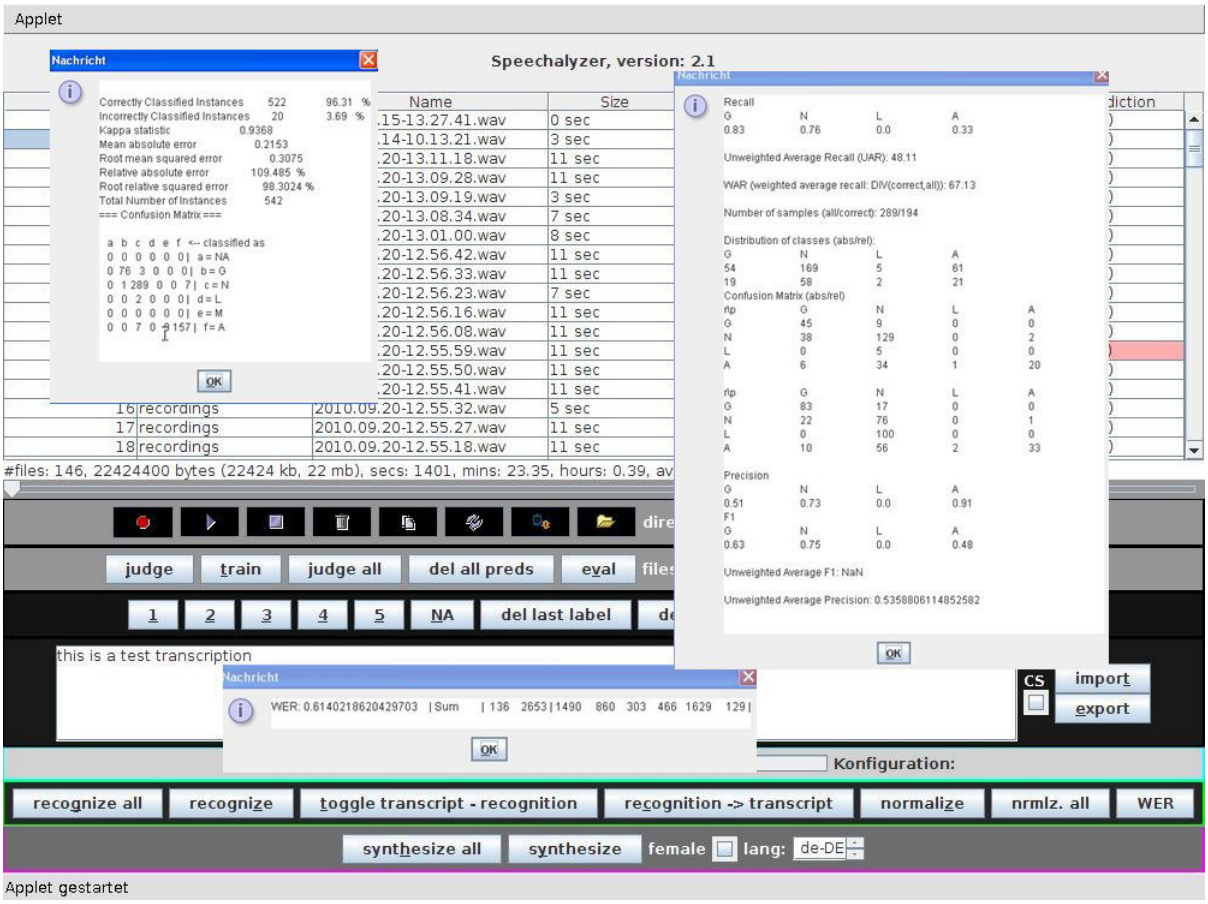

Figure 3 - Displaying evaluation output with the Speechalyzer client GUI, a 10 fold cross evaluation, the word error rate and classification statistics.

[3] BURKHARDT, F., K. ENGELBRECHT, M. VAN BALLEGOOY, T. POLZEHL and J. STEGMANN:: *Emotion Detection in Dialog Systems - Usecases, Strategies and Challenges*. In *Proceedings Affective Computing and Intelligent Interaction (ACII), Amsterdam, The Netherlands*, 9 2009.

[4] BURKHARDT, F., T. POLZEHL, J. STEGMANN, F. METZE and R. HUBER: *Detecting real life anger*. In *Proceedings ICASSP, Taipei; Taiwan*, 4 2009.

[5] EYBEN, F., M. WÖLLMER and B. SCHULLER: *openSMILE – The Munich Versatile and Fast Open-Source Audio Feature Extractor*. In *Proc. of ACM Multimedia*, pp. 1459–1462, Florence, Italy, October 2010. ACM.

[6] HÖGE, H., C. DRAXLER, H. VAN DEN HEUVEL, F. T. JOHANSEN, E. SANDERS and H. S. TROPF: *SpeechDat Multilingual Speech Databases for Teleservices: Across the Finish Line*. In *Proc. of Eurospeech*, 1999.

[7] HUNSPELL: *http://hunspell.sourceforge.net/*.

[8] PALLET, D.: *A Look at NIST's Benchmark ASR tests: Past, Present, and Future*. Proc. Automatic Speech Recognition and Understanding (ASRU), 2003.

[9] SCHRÖDER, M., P. BAGGIA, F. BURKHARDT, A. OLTRAMARI, C. PELACHAUD, C. PETER and E. ZOVATO: *W3C Emotion Markup Language (EmotionML) 1.0*. http://www.w3.org/TR/emotionml/, 2010.

[10] SCHULLER, B., S. STEIDL, A. BATLINER, F. BURKHARDT, L. DEVILLERS, C. MÜLLER and S. NARAYANAN: *The INTERSPEECH 2010 Paralinguistic Challenge*. In *Proc. Interspeech*, Makuhari, Japan, 2010.

[11] SJÖLANDER, K. and J. BESKOW: *Wavesurfer - An Open Source Speech Tool*, 2000.

[12] WITTEN, I. H. and E. FRANK: *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Second ed., June 2005.