TRANSCRATER: A TOOL FOR AUTOMATIC SPEECH RECOGNITION QUALITY ESTIMATION

VERSION: 1.1

Authors

Shahab Jalalvand

Matteo Negri

Daniele Falavigna

Marco Turchi

Mohammed R H Qwaider

July 2016

Contents

1 Introduction to TranscRater

An open-source tool for automatic speech recognition (ASR) quality estimation (QE). The tool allows performing ASR evaluation bypassing the need of reference transcripts and confidence information, which is common to current assessment protocols. TranscRater includes: *i)* methods to extract a variety of quality indicators from (*signal, transcription*) pairs and *ii)* machine learning algorithms which allow building ASR QE models exploiting the extracted features.
TranscRater answers the key problem of *How to determine the quality of an automatic transcription without reference transcripts and without confidence information*? To do so it provides:

- an accurate method to estimate ASR output quality at run-time
- a method to predict ASR output quality that is reference-independent
- a method to predict ASR output quality that is also confidence-independent
- an efficient method for quality-based ranking when multiple transcriptions are available

TranscRater has been developed by Shahab Jalalvand, PhD student at the university of Trento and it includes the contributions from a number of researchers mainly Matteo Negri, Marco Turchi, Jose José G. C. de Souza and Daniele Falavigna from FBK (Fondazione Bruno Kessler) research center [Negri et al.2014, C. de Souza et al.2015, Jalalvand et al.2015b, Zamani et al.2015].

1.1 Applications

When the costly manual transcriptions are not available and you need to evaluate the performance of the ASR output, then you can use TranscRater to predict the quality of transcriptions. Moreover, if you have several transcription channels like multiple microphones or multiple ASR systems, then you can use this toolkit to compare and rank performances.
This toolkit will be useful for any task that uses ASR output. The effective performance of TranscRater has been already reported in several publications:

- in [Negri et al.2014] to predict the WER in absence of ASR decoder features
- in [Jalalvand et al.2015b] to reorder the input of ROVER (an ASR system combination)
- in [Jalalvand et al.2015a] to select the most suitable adaptation data for DNN-HMM based acoustic model adaptation

2 User's Guide

2.1 System requirements

The user needs to have python 2.7 and java 1.8 already installed on its station. Moreover, the folowing python libraries are needed:

- "json"
- "SciPy"

- "yaml"
- "sklearn" python library
  (http://scikit-learn.org/stable/install.html)
- "python_speech_features" for signal processing toolkit
  (https://github.com/jameslyons/python_speech_features)

The auxiliary tools to be compiled are:

- "RNNLM" recurrent neural network language model toolkit
  (http://www.fit.vutbr.cz/ imikolov/rnnlm/rnnlm-0.3e.tgz)
- "SRILM" n-gram language model toolkit
  (http://www.speech.sri.com/projects/srilm/download.html)
- "TreeTagger" part-of-speech tagging toolkit
  (http://www.cis.uni-muenchen.de/ schmid/tools/TreeTagger/data/tree-tagger-linux-3.2.tar.gz)
- "RankLib" v2.6
  (https://sourceforge.net/projects/lemur/files/lemur/RankLib-2.6/RankLib-2.6.jar/download)

The paths for the required tools are to be set in config.json'.
RNNLM is a toolkit to train and test the recurrent neural network language models. It can be freely downloaded from:
`http://www.fit.vutbr.cz/~imikolov/rnnlm/rnnlm-0.3e.tgz`
We will use the "rnnlm" executable file to calculate the probability of the words. After compiling the toolkit, the user needs to set the path to this function in config.json'. Once installed/compiled, one can train his language model using the following command.

```
$RNNLMDIR/rnnlm -train train.txt -valid valid.txt
-rnnlm ./models/rnnlm1
```

SRILM is a toolkit to train and test the n-gram language models. It can be freely downloaded from
`http://www.speech.sri.com/projects/srilm/download.html`
We will use the executable file named "ngram". After installation, the user needs to set the path to this function in config.json. One can train his 4-gram language model by the following command:

```
$SRILMDIR/ngram-count -text train.txt -order 4 -lm ./models/4grLM1 -gt3min
1 -gt4min 1 -kndiscount -interpolate -unk
```

TreeTagger is a tool for annotating text with part-of-speech and lemma information. It can be freely downloaded from:

```
http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger-linux-3
2.tar.gz
```

We will use the executable file named "tree-tagger", so that the user is asked to set the path to this function in config.json.
RankLib is a library for running machine-learned ranking algorithms. It can be freely downloaded from:

```
https://sourceforge.net/projects/lemur/files/lemur/RankLib-2.6/RankLib-2.
6.jar/download
```

Once downloaded, the user needs to set the path "RankLib-2.6.jar" in config.json. In this toolkit we only use RandomForest as the ranking machine as it has been shown to be efficient for ASR-QE. However in RankLib there other types of machines available.

2.2 Usage

Once you have all the aforementioned requirements, then you can start using TrancRater.

2.2.1 Fast run

In order to simply train the QE models, skipping all the feature extraction phases, we have provided a train and test sample data.
In the root folder run the following command:
`TranscRater> python fast_run_RR.py`
This trains a regression model on "./data/train_SIG_LEX.data" and test it on "./data/test_SIG_LEX.data". The process takes 59 seconds on my PC with 8 processors 3.4Ghz and 8 Gig RAM. The output looks like Figure 1.

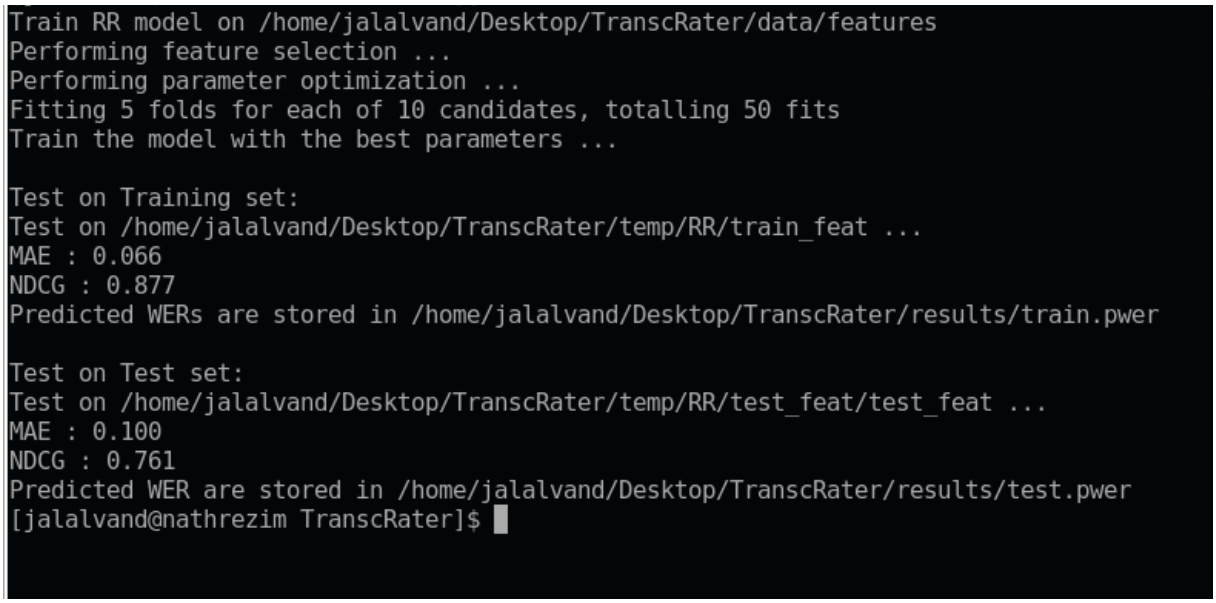

Figure 1: Train and test RR models.

The first MAE (the lower the better) and NDCG (the higher the better) numbers, respectively, show WER prediction and ranking prediction performance on the training data. The second MAE and NDCG values show the performances on test data.
Again in the root directory, one can run the following command to use machine-learned ranking (MLR) solution:
`TranscRater> python fast_run_MLR.py`
This script trains and tests the MLR models. The output looks like Figure 2.
The whole procedure takes 11 seconds. The first NDCG number shows the ranking performance on the training data and the second number shows the performance on the test data.
Comparing the NDCG values (0.761 v.s. 0.865) shows that the better ranking results can be obtained by learning to rank approaches.

Figure 2: Train and test MLR models.

2.2.2 Configuration file

Figure 3 shows the configuration parameters that are needed to be set by the user.
The toolkit makes some temporary folders in the base directory to store the temporary files and the results.

2.2.3 Input files

TranscRater uses two sets of inputs: audio data and transcription texts.
The format of the audio files is as:
`RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 16000 Hz`
The format of the transcription files is as:
`utterance-id <sentence>`
and it is the same as the format of the reference files. **Note that the utterance ids should appear with the same order in references and transcriptions and the audio lists**.
Also note that for SIG features, one need to provide the transcriptions in .ctm format (i.e. including the time boundary of the recognized words).

2.2.4 Output files

For each test sample, the output is either a predicted WER or a predicted rank wrt to the other samples and reliability of these predictions can be respectively evaluated in terms of MAE or NDCG. Output predictions are provided in a single file (one WER prediction per row for regression and one rank prediction per row for ranking).

3 Developer's Guide

TranscRater consists of two main modules: feature extraction and machine learning. Figure 4 shows these modules and their core scripts.

```
{
"BASEDIR":"the framework directory",
"BINDIR":"the TranscRater directory",

"RNNLMDIR" : "the RNNLM directory",
"SRILMDIR" : "the SRILM directory",
"TREETAGDIR" : "the TreeTagger directory",
"RANKLIBDIR" : "the RankLib directory",

"RNNLM1" : "the first RNNLM",
"RNNLM2" : "the second RNNLM",
"SRILM1" : "the first n-gram LM",
"SRILM2" : "the second n-gram LM",

"LEXFEAT" : "the Lexicon feature dictionary",

"CHANNELS" : "number of transcription channels",

"trainREF" : "training reference",
"train_waveChannels" : "the training list files of the speech signals",
"train_transcChannels" : "the training transcription files",
"train_transcChannels_ctm" : "the training transcription files (ctm format)",

"testREF" : "test reference",
"test_waveChannels" : "the test list files of the speech signals",
"test_transcChannels" : "the test transcription files",
"test_transcChannels_ctm" : "the test transcription files (ctm format)",

"MICROPHONES":"number of recording channels",
"ASR_SYSTEMS":"number of ASR systems",

"FEAT" : "the type of features",

"folds":"number of folds for tuning",
```

Figure 3: The configuration file.

3.1 Feature Extraction

In this section we introduce the functions and scripts to perform feature extraction on the training set. The same procedure works for the test set as well.
Here we assume that in a python console like ipython the configuration file ("config.json") is loaded by:

```
>>> with open("config.json", 'r') as f:
config = json.load(f)
```

3.1.1 Labels

When the reference file is given in "data/train.ref" and the automatic transcriptions are given in "data/transcriptions/train_C
then the WER of each transcription is computed using the following command:

```
>>> import get_WER_scores
>>> get_WER_scores.main("train")
```

The output files are stored in "data/features/train_CH_*.wer". The WERs will be used as the label for regression algorithms. The WERs are also used to derive the ranks in "data/features/train_CH_*.rank" as the labels for the ranking machines.

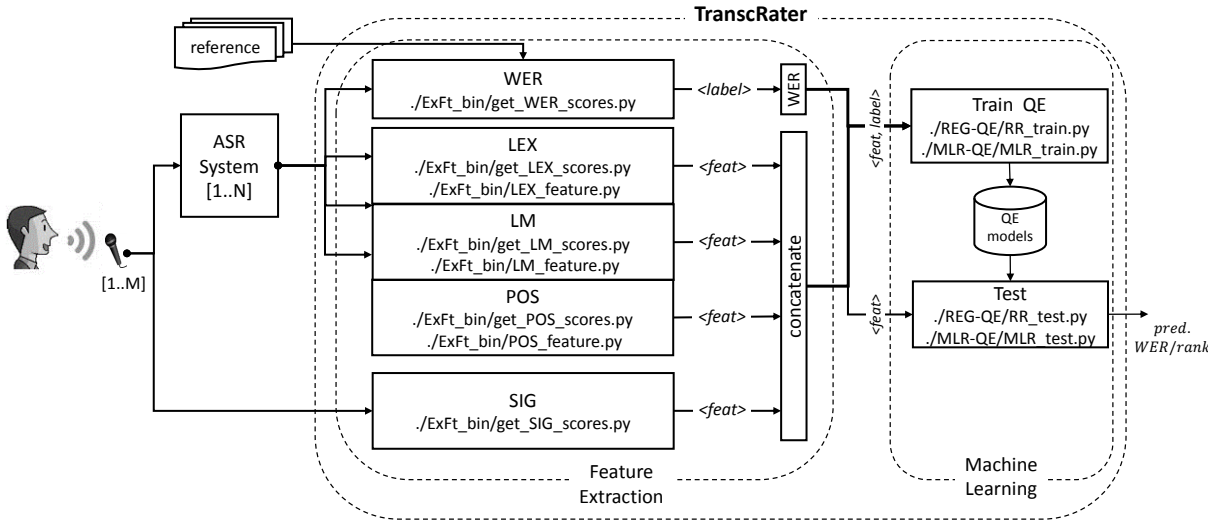

Figure 4: The structure of TranscRater.

3.1.2 features

For the transcriptions given in "train_transcChannels" variable, the toolkit extracts three types of features: LEX, LM and POS.

```
>>> import get_LEX_features
>>> get_LEX_scores.main("train")

>>> import get_LM_features
>>> get_LM_scores.main("train")

>>> import get_POS_features
>>> get_POS_scores.main("train")
```

From the audio files given in "train_wavChannels" and the ctm transcription files given in "train_transcChannels_ctm", the tool computes the SIG features.

```
>>> import get_SIG_features
>>> get_SIG_scores.main("train")
```

The output of the feature extraction module will be stored in "./data/features/" directory:

```
./data/features/train_CH_*_LEX.feat
./data/features/train_CH_*_LM.feat
./data/features/train_CH_*_POS.feat
./data/features/train_CH_*_SIG.feat
```

Each line contain the feature vector for one transcription.
In addition to these feature types, the user can add his own that we call them user defined features (UDF). These features must be stored in "./data/feature/train_CH_i_UDF.feat" file and they must follow the same format of the other feature files, i.e. in terms of the number of lines and each line being a vector of float

numbers.
`./data/features/train_CH_*_UDF.feat`
The same set of commands must be run on the test set, simply by changing all the "train" terms into "test".

3.2 Machine Learning

3.2.1 Regression (RR)

When the feature files are ready, then the following command will use them along with the label files to train the regression models and to save the models into "RR_models" directory.

```
>>> import RR_train
>>> RR_train.main("data/features/", "RR_models")
```

To test the RR models, one can use the following command:

```
>>> import RR_test
>>> RR_test.main("data/features/", "RR_models", "results/")
```

3.2.2 Machine-learned ranking (MLR)

In case of having multiple transcription channels, one can use TranscRater to rank the multiple transcriptions based on their quality using learning to rank algorithms.
The following command uses the features and ranks, previously computed in "./data/features/" to prepare the data for training MLR models:

```
>>> import MLR_data
>>> MLR_data.main("train")
```

This makes the training data with the format of SVM light which is also used by RankLib library.
The resulting file will be stored in:
`./data/MLR_train_FEATURE.data`
"FEATURE" represents the types of the desired features in config.json.
The following command trains the MLR models and stores them in "./MLR_models" directory:

```
>>> import MLR_train
>>> MLR_train.main("./data/MLR_train_FEATURE.data", "MLR_models"
```

For test, we need to first prepare the data:

```
>>> import MLR_data
>>> MLR_data.main("test")
```

and then predict the ranks using MLR models:

```
>>> import MLR_test
>>> MLR_test.main("./data/MLR_test_FEATURE.data", "MLR_models", "results/test.prar
```

4 Example on CHiME3

For this example, we use the data from the 3^{rd} CHiME challenge,[1] which were collected for multiple distant microphone speech recognition in noisy environments [Barker et al.2015]. CHiME-3 data consists of sentences of the Wall Street Journal corpus, uttered by four speakers in four noisy environments,

[1] `http://spandh.dcs.shef.ac.uk/chime_challenge/chime2015/data.html`

and recorded by five frontal microphones placed on the frame of a tablet PC (a sixth one, placed on the back, mainly records the background noise).
In this example we are interested in predicting the quality of each microphone and then ranking them using 1) the RR method and 2) the MLR method.
Training and test sets respectively contain 1,640 and 1,320 sentences. Transcriptions are produced by a baseline ASR system, provided by the task organizers, which uses the deep neural network recipe of Kaldi [Povey et al.2011].
The transcriptions of the frontal microphones (1st, 3rd, 4th, 5th and 6th) using the baseline ASR system are also provided in:

```
./data/transcriptions/train_CH_1.txt (.ctm)
./data/transcriptions/train_CH_2.txt (.ctm)
./data/transcriptions/train_CH_3.txt (.ctm)
./data/transcriptions/train_CH_4.txt (.ctm)
./data/transcriptions/train_CH_5.txt (.ctm)
./data/transcriptions/test_CH_1.txt (.ctm)
./data/transcriptions/test_CH_2.txt (.ctm)
./data/transcriptions/test_CH_3.txt (.ctm)
./data/transcriptions/test_CH_4.txt (.ctm)
./data/transcriptions/test_CH_5.txt (.ctm)
```

The references for the training and test sets are provided in:

```
./data/train.ref
./data/test.ref
```

4.1 Prepare audio data

In order to use the signal based features, the user needs to download the audio data from `http://spandh.dcs.shef.ac.uk/chime_challenge/chime_download.html` (that is freely available) and he needs to provide a list of the audio recodings (full path) in a file and put the name of these files in the corresponding field in the configuration file:

```
"train_waveChannels":
"test_waveChannels":
```

To do so:

1. click on the link above
2. download "CHiME3_isolated_dt05_real"
3. select "CHiME3_isolated_dt05_real.zip" option and click on download
4. unzip the file and save the path to this folder in a variable like "CHDIR"

Follow the same procedure to download the evaluation set by selecting "CHiME3_isolated_et05_real" and then "CHiME3_isolated_et05_real.zip". As mentioned before, the transcriptions of these audio files are provided in "./data/transcriptions/test_CH_*.txt (.ctm)".
After downloading the audio files the user needs to prepare a list file for each microphone, in the same order as the reference files. For example, if the unzipped file is in "CHDIR=/home/jalalvand/Downloads/CHiME3", then one can use the following bash command to make the audio list file for each microphone:

```
TranscRater> for Mic in 1 3 4 5 6; do
```

```
cat data/train.ref | cut -d' ' -f1 |
sed "s/_real//g" | tr '[:lower:]' '[:upper:]' |
while read id; do
find ${CHDIR}/data/audio/16kHz/isolated/dt05_*/${id}.CH${Mic}.wav
done > ./data/lists/train_CH_${Mic}_wav.list
```

Note that we won't use the second microphone as it is placed on the back side of the tablet device and it mostly captures the noise.

The same command can be used for the evaluation set:

```
TranscRater> for Mic in 1 3 4 5 6; do
cat data/test.ref | cut -d' ' -f1 |
sed "s/_real//g" | tr '[:lower:]' '[:upper:]' |
while read id; do
find ${CHDIR}/data/audio/16kHz/isolated/et05_*/${id}.CH${Mic}.wav
done > ./data/lists/test_CH_${Mic}_wav.list
```

4.2 Configuration File

The configuration file is set as in Figure 5. The user has to change some of the variables according to the local values.

```
{"BASEDIR":"/home/jalalvand/Desktop/TranscRater/egs/CHiME3",
"BINDIR":"/home/jalalvand/Desktop/TranscRater",

"RNNLMDIR" : "/home/jalalvand/Desktop/rnnlm/rnnlm-0.3e",
"SRILMDIR" : "/home/jalalvand/Desktop/srilm/bin/i686-m64",
"TREETAGDIR" : "/home/jalalvand/Desktop/fbk_functions/TreeTagger/bin",
"RANKLIBDIR" : "/home/jalalvand/Downloads",
"LEXFEAT" : "/home/jalalvand/Desktop/TranscRater/AUXFILE/LexicalFeatures.txt",

"RNNLM1" : "",
"RNNLM2" : "",
"SRILM1" : "lm/srilm1M_4gr.blm",
"SRILM2" : "",

"CHANNELS" : "5",

"trainREF" : "data/train.ref",
"train_waveChannels" : "data/lists/train_CH_1_wav.list data/lists/train_CH_2_wav.list
data/lists/train_CH_3_wav.list data/lists/train_CH_4_wav.list data/lists/train_CH_5_wav.list",
"train_transcChannels" : "data/transcriptions/train_CH_1.txt data/transcriptions/train_CH_2.txt
data/transcriptions/train_CH_3.txt data/transcriptions/train_CH_4.txt
data/transcriptions/train_CH_5.txt",
"train_transcChannels_ctm" : "data/transcriptions/train_CH_1.ctm
data/transcriptions/train_CH_2.ctm data/transcriptions/train_CH_3.ctm
data/transcriptions/train_CH_4.ctm data/transcriptions/train_CH_5.ctm",

"testREF":"data/test.ref",
"test_waveChannels" : "data/lists/test_CH_1_wav.list data/lists/test_CH_2_wav.list
data/lists/test_CH_3_wav.list data/lists/test_CH_4_wav.list data/lists/test_CH_5_wav.list",
"test_transcChannels" : "data/transcriptions/test_CH_1.txt data/transcriptions/test_CH_2.txt
data/transcriptions/test_CH_3.txt data/transcriptions/test_CH_4.txt
data/transcriptions/test_CH_5.txt",
"test_transcChannels_ctm" : "data/transcriptions/test_CH_1.ctm
data/transcriptions/test_CH_2.ctm data/transcriptions/test_CH_3.ctm
data/transcriptions/test_CH_4.ctm data/transcriptions/test_CH_5.ctm",

"MICROPHONES":"5",
"ASR_SYSTEMS":"1",
"FEAT" : "LEX_POS",
"folds":"5",
"RR_Iter":"10",
"n_jobs" : "1",
"MLR_Tune":"0"}
```

Figure 5: The configuration file for CHiME3 example.

With this configuration file, running the following command:

```
TrnascRater> python run-QE.sh
```

1. computes the WER of each channel on each sentence
2. extracts the LEX features
3. extracts the LM features
4. extracts the POS features
5. does the same for test set
6. trains the RR models and shows the corresponding MAE and NDCG
7. predicts the WERs of the test sets and shows the MAE and NDCG on test set
8. trains MLR models and shows the NDCG score on the training set
9. predicts the ranks using MLR models and shows the NDCG score on the test set

References

[Barker et al.2015] Jon Barker, Ricard Marxer, Emmanuel Vincent, and Shinji Watanabe. 2015. The third 'CHiME' Speech Separation and Recognition Challenge: Dataset, Task and Baselines. In *Proc. of the 15th IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 1–9, Scottsdale, Arizona, USA.

[C. de Souza et al.2015] José G. C. de Souza, Hamed Zamani, Matteo Negri, Marco Turchi, and Daniele Falavigna. 2015. Multitask Learning for Adaptive Quality Estimation of Automatically Transcribed Utterances. In *Proc. of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*, pages 714–724, Denver, Colorado, USA.

[Jalalvand et al.2015a] Shahab Jalalvand, Daniele Falavigna, Marco Matassoni, Piergiorgio Svaizer, and Maurizio Omologo. 2015a. Boosted Acoustic Model Learning and Hypotheses Rescoring on the CHiME-3 Task. In *Proc. of the IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 409–415, Scottsdale, Arizona, USA.

[Jalalvand et al.2015b] Shahab Jalalvand, Matteo Negri, Daniele Falavigna, and Marco Turchi. 2015b. Driving ROVER With Segment-based ASR Quality Estimation. In *Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1095–1105, Beijing, China.

[Negri et al.2014] Matteo Negri, Marco Turchi, José G. C. de Souza, and Falavigna Daniele. 2014. Quality Estimation for Automatic Speech Recognition. In *Proc. of the 25th International Conference on Computational Linguistics: Technical Papers (COLING)*, pages 1813–1823, Dublin, Ireland.

[Povey et al.2011] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. 2011. The Kaldi Speech Recognition Toolkit. In *Proc. of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Hawaii, USA.

[Zamani et al.2015] Hamed Zamani, José GC de Souza, Matteo Negri, Marco Turchi, and Daniele Falavigna. 2015. Reference-free and Confidence-independent Binary Quality Estimation for Automatic Speech Recognition. In *Proceedings of the Second Italian Conference on Computational Linguistics CLiC-it 2015*, Trento, Italy, December.