

TranscRater: a Tool for Automatic Speech Recognition Quality Estimation

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, Italy 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.2015, 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 its quality. Moreover, if you have several transcription channels like multiple microphones or multiple ASR systems, then you can use this toolkit to compare and rank their performances.

For any task that uses ASR output this toolkit will be useful. Spoken translation, multiple distant ASR systems and ASR system combination (such as ROVER) are some of these tasks [Jalalvand et al.2015].

2 User's Guide

2.1 System requirements

On a Linux system, the machine learning module requires:

- "java" 8 (<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>)
- "Python" v2.7
- "NumPy" 1.6.1 and "SciPy" 0.9
- "sklearn" python library (<http://scikit-learn.org/stable/install.html>)
- "RankLib" v2.6 (<https://sourceforge.net/projects/lemur/files/lemur/RankLib-2.6/RankLib-2.6.jar/download>)

The requirments for feature extraction:

- "OpenSmile" signal processing toolkit
- "RNNLM" recurrent neural network language model toolkit
- "SRILM" n-gram language model toolkit
- "TreeTagger" part-of-speech tagging toolkit

The paths for the required tools and libraries are to be set in "configuration.conf". "OpenSmile" is a fast signal feature extraction tool. It can be freely downloaded from

<http://www.audeering.com/research/opensmile#download>

We will use the executable file named “SMILExtract” to extract the signal features, therefore the user is asked to set the variable ”OPENSMAILEDIR=” referring to the path of this file in the “configuration.conf” file.

“**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. The user is asked to set the path to this function in the “configuration.conf” file after compiling the toolkit. Once you install/compile RNNLM toolkit, you can train your own 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” so that the user is asked to set the path to this function in the “configuration.conf” file after installing the toolkit. Once you install/compile the SRILM toolkit, you can train your own 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 “configuration.conf”.

“**RankLib**” is a library for machine-learned ranking (or learning to rank) algorithms. It can be freely downloaded from

<https://sourceforge.net/projects/lemur/files/lemur/RankLib-2.6/RankLib-2.6.jar/download>

Once you download and extract the folder, you must put the path to the “RankLib-2.6.jar” file in the “configuration.conf” file. In this toolkit we only use Random-Forest as the ranking machine as it has been shown to be the best for ASR-QE. However in RankLib there other types of machines available.

2.2 Usage

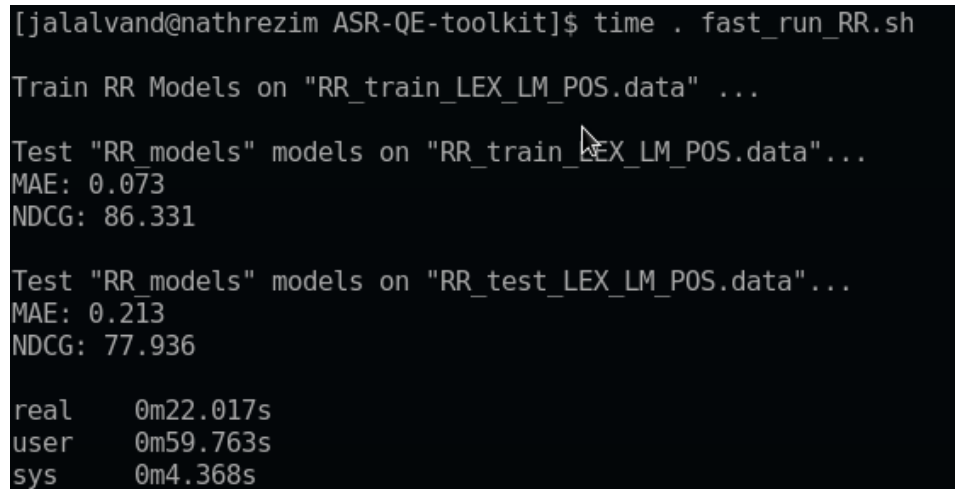
Once you have all the aforementioned requirements then you can start using TransRater.

2.2.1 First 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 as well as a simple bash script. In the root folder run the following command:

```
. fast_run_RR.sh
```

This trains a regression model on "train_LEX_LM_POS.data" and test it on "test_LEX_LM_POS.data". The process takes 22 seconds on my PC with 8 processors 3.4Ghz and 8 Gig RAM. The output looks like Figure 1.



```
[jalalvand@nathrezim ASR-QE-toolkit]$ time . fast_run_RR.sh

Train RR Models on "RR_train_LEX_LM_POS.data" ...

Test "RR models" models on "RR_train_LEX_LM_POS.data"...
MAE: 0.073
NDCG: 86.331

Test "RR models" models on "RR_test_LEX_LM_POS.data"...
MAE: 0.213
NDCG: 77.936

real    0m22.017s
user    0m59.763s
sys     0m4.368s
```

Figure 1: Train and test RR models.

The first MAE (the lower the better) and NDCG (the higher the better) numbers, respectively, show the WER prediction and ranking prediction performances on the training data. The second numbers, respectively, show the performances on test data.

Again in the root directory one can run the following command:

```
. fast_run_MLR.sh
```

to train and test the MLR models. The output looks like Figure 2.

The whole procedure takes 32 second. The first NDCG number shows the ranking performance on the training data and the second number shows the performance on the test data.

2.2.2 Configuration file

Figure 3 shows the configuration parameters that need to be set by the user.

```
[jalalvand@nathrezim ASR-QE-toolkit]$ time . fast_run_MLR.sh
Make 5 fold data...done

Train MLR models on "MLR_train_LEX_LM_POS.data"...done

Test "MLR models" on "MLR_train_LEX_LM_POS.data"...
NDCG: 85.540

Test "MLR models" on "MLR_test_LEX_LM_POS.data"...
NDCG: 80.793

real    0m32.213s
user    0m42.781s
sys     0m5.001s
```

Figure 2: Train and test MLR models.

Note that the stored fields are the mandatory ones. Also the toolkit makes some temporary folders in the base directory to store the temporary files and the results. See Figure 3 to know the needed configuration fields.

2.2.3 Input files

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 reference files is as:

utterance-id <sentence>

and it is the same as the format of the transcription files. **Note that the utterance ids should appear with the same order in both reference and transcriptions.**

The list of the audio files should also follow the same order.

Internally, TranscRater stores the extracted features in the SVM-light¹ format. This makes possible to use the tool as a feature extractor and to embed it in applications different from the ones described in this manual.

2.2.4 Output files

For each test point, the output is either a WER prediction or a rank, whose reliability 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

¹<http://svmlight.joachims.org/>

```

BASEDIR*= the work-space directory
BINDIR*=  the root directory of the toolkit

OPENSIMLEDIR= the directory of OpenSmile lib
RNNLMDIR=     the directory of RNNLM
SRILMDIR=     the directory of SRILM
TREETAGDIR=   the directory of TreeTagger
RANKLIBDIR=   the directory of RankLib

RNNLM1= the out-domain RNNLM
RNNLM2= the in-domain RNNLM
SRILM1= the out-domain SRILM
SRILM2= the in-domain SRILM

LEXFEAT= the lexical feature dictionary

trainREF*= the training reference
train_wavChannels= the training audio files
train_transcChannels*=the training auto transcriptions

testREF= the test references
test_wavChannels= the test audio files
test_transcChannels*= the test auto transcriptions

MICROPHONES*= the number of microphones
ASR_SYSTEMS*= the number of ASR systems

folds*= the number of folds

QE*=RR "RR" for regression methods; "MLR" for rank
RR_Iter=20 the number of training iterations for RR me
MLR_Tune=No set "Yes" if you want to tune on training

* at least one the feature typse must be set to 1
F_SIG=0 set "1" if you want to use signal features
F_LEX=1 set "1" if you want to use lexical features
F_LM=0 set "1" if you want to use language model feature
F_POS=0 set "1" if you want to use part-of-speech based :

```

Figure 3: The configuration file.

rank prediction per row for ranking). MAE or NDCG scores are provided as the standard output of the test functions.

3 Developer's Guide

TranscRater consists of two main modules: feature extraction and machine learning. Figure 5 shows these modules and their core scripts.

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 test set as well.

```

a) ---RR file format Single Channel-----
0.429 qid:01 1:5.01 2:9.58 3:-4.49 ... #CH_1
0.286 qid:02 1:7.47 2:5.87 3:3.39 ... #CH_1
0.700 qid:03 1:5.92 2:9.02 3:0.19 ... #CH_1
-----

---RR file format Multiple Channel-----
0.429 qid:01 1:5.01 2:9.58 3:-4.49 ... #CH_1
0.357 qid:01 1:5.01 2:9.59 3:-2.57 ... #CH_2
0.571 qid:01 1:5.01 2:-2.30 3:1.09 ... #CH_3
b) 0.286 qid:02 1:7.47 2:5.87 3:3.39 ... #CH_1
0.286 qid:02 1:7.47 2:3.95 3:5.41 ... #CH_2
0.357 qid:02 1:7.47 2:1.15 3:-3.67 ... #CH_3
0.700 qid:03 1:5.92 2:9.02 3:0.19 ... #CH_1
0.550 qid:03 1:5.92 2:10.28 3:-0.40 ... #CH_2
0.500 qid:03 1:5.92 2:6.13 3:-1.49 ... #CH_3
-----

c) ---MLR file format Multiple Channel-----
2 qid:01 1:5.01 2:9.58 3:-4.49 ... #CH_1
1 qid:01 1:5.01 2:9.59 3:-2.57 ... #CH_2
3 qid:01 1:5.01 2:-2.30 3:1.09 ... #CH_3
1 qid:02 1:7.47 2:5.87 3:3.39 ... #CH_1
1 qid:02 1:7.47 2:3.95 3:5.41 ... #CH_2
2 qid:02 1:7.47 2:1.15 3:-3.67 ... #CH_3
3 qid:03 1:5.92 2:9.02 3:0.19 ... #CH_1
2 qid:03 1:5.92 2:10.28 3:-0.40 ... #CH_2
1 qid:03 1:5.92 2:6.13 3:-1.49 ... #CH_3
-----

```

Figure 4: The format of the internal data files.

3.1.1 Label computation

When the reference file is given in "train.ref" and the automatic transcription is given in train.hyp, then the WER of each transcription is computed using the following command:

```
./ExFt_bin/get_WER_scores.sh train.ctm train.stm train.wer
```

"train.wer" is output file that includes the WER of each transcription and will be used as the label for regression algorithms. Also it will be used to derive the ranks as the labels for the ranking machines.

3.1.2 feature vectores

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

```
./ExFt_bin/get_LEX_features.sh train
./ExFt_bin/get_LM_features.sh train
```

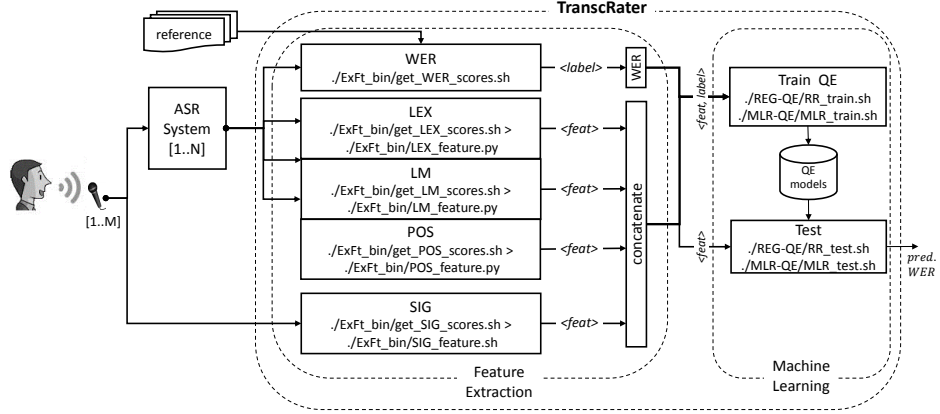


Figure 5: The structure of TranscRater.

```
./ExFt_bin/get_POS.features.sh train
```

From the audio files given in "train_wavChannels", the tool computes the SIG features

```
./ExFt_bin/get_SIG.features.sh train
```

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

```
./data/features/train.CH.1.LEX.feats
```

```
./data/features/train.CH.1.LM.feats
```

```
./data/features/train.CH.1.POS.feats
```

```
./data/features/train.CH.1.SIG.feats
```

The same for the other transcription channels (CH.2, CH.3, so on) if they exist.

3.2 Machine Learning

3.2.1 Regression (RR)

When the feature files are ready, then the following command will prepare the data file for training regression models:

```
./REG-QE/RR_data.sh train
```

According to the feature types and the number of channels defined in configuration file and using the WER scores computed beforehand, this command will prepare a data file in SVM.Light format in:

```
./data/RR.train.FEATURE.data
```

FEATURE can be SIG, LEX, LM, POS or any combination of these values.

Afterwards the following command trains the regression models on "RR_train_FEATURE.data" and stores them in "RR_models":

```
./REG-QE/RR_train.sh RR_train_FEATURE.data k RR_models
```

where "k" is the number of folds.

To test the RR models, one can use the following commands to first prepare the test data:

```
./REG-QE/RR_data.sh test
```

and then to predict the WER scores:

```
./REG-QE/RR_test.sh RR_test_FEATURE.data RR_models RR_test.pwer
```

3.2.2 Machine-learned ranking (MLR)

In case of having multiple transcription channels, one can use TranscRater to rank different transcriptions based on their quality using learning to rank algorithms.

The following command uses the features and ranks, previously computed in "/data/features/" and prepare the data for training MLR models:

```
./MLR-QE/MLR_data.sh train
```

The resulting file will be stored in:

```
./data/MLR_train_FEATURE.data
```

The following command trains the MLR models and stores them in "MLR_models" directory:

```
./MLR-QE/MLR_train.sh MLR_train_FEATURE.data k MLR_models
```

For test, we need to first prepare the data:

```
./MLR-QE/MLR_data.sh test
```

And then predict the ranks using MLR models:

```
./MLR-QE/MLR_test.sh MLR_test_FEATURE.data MLR_models MLR_test.prank
```

4 Complete Example on CHiME3

For this example, we use the data from the 3rd CHiME challenge,² 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, and recorded by five frontal microphones placed on the frame of a tablet PC (a sixth one, placed on

²http://spandh.dcs.shef.ac.uk/chime_challenge/chime2015/data.html

the back, mainly records background noise). Training and test 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
./data/transcriptions/train_CH.3.txt
./data/transcriptions/train_CH.4.txt
./data/transcriptions/train_CH.5.txt
./data/transcriptions/train_CH.6.txt
./data/transcriptions/test_CH.1.txt
./data/transcriptions/test_CH.3.txt
./data/transcriptions/test_CH.4.txt
./data/transcriptions/test_CH.5.txt
./data/transcriptions/test_CH.6.txt
```

The “./egs/CHiME3” directory includes the reference for for the training and test sets:

```
./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 provides a list of the audio files (full path) in a file and put the name of this list in the corresponding field in the configuration file.

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 ”CHDIR” variable

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 by the baseline ASR systems are provided in “./data/transcriptions/test_CH.i.txt”.

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}.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}.list
```

4.2 Configuration File

The configuration file is set as in Figure 6. The user has to change some of the variables according to the local values.

With this configuration file, the following command:

```
TranscRater> . ../bin/run-QE.sh configuration1.conf
```

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 MAP
7. predicts the WERs of the test sets and shows the MAE and MAP on test set

By changing the variable “QE=RR” into “QE=MLR”, instead of regression models, the toolkit prepares the ranking models resulting better MAP measure.

```

# !/bin/bash Configuration paths for CHiME3
# ----- Framework DIRs
BASEDIR=/home/jalalvand/Desktop/ASR-QE-toolkit/egs/CHiME3
BINDIR=/home/jalalvand/Desktop/ASR-QE-toolkit
# ----- Library and Tool DIRs
OPENSMLIEDIR=/home/jalalvand/Desktop/toolkits/opensmile-2.0-rc1/opensmile
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/Desktop/toolkits/RankLib-v2.1/bin
# ----- Language model DIRs
#RNNLM1=$BASEDIR/lm/rnnlm1M
#RNNLM2=$BASEDIR/lm/rnnlm1M
#SRILM1=$BASEDIR/lm/srilm37M_4gr.blm
#SRILM2=$BASEDIR/lm/srilm1M_4gr.blm
# ----- Auxiliary files
LEXFEAT=$BINDIR/AUXFILE/LexicalFeatures.txt
# ----- Train Channels
trainREF=$BASEDIR/data/train.ref
train_wavChannels=($BASEDIR/data/lists/train_Mic_1.list
$BASEDIR/data/lists/train_Mic_3.list
$BASEDIR/data/lists/train_Mic_4.list
$BASEDIR/data/lists/train_Mic_5.list
$BASEDIR/data/lists/train_Mic_6.list)
train_transcChannels=($BASEDIR/data/transcriptions/train_CH_1.txt
$BASEDIR/data/transcriptions/train_CH_3.txt
$BASEDIR/data/transcriptions/train_CH_4.txt
$BASEDIR/data/transcriptions/train_CH_5.txt
$BASEDIR/data/transcriptions/train_CH_6.txt)
# ----- Test Channels
testREF=$BASEDIR/data/test.ref
test_wavChannels=($BASEDIR/data/lists/test_Mic_1.list
$BASEDIR/data/lists/test_Mic_3.list
$BASEDIR/data/lists/test_Mic_4.list
$BASEDIR/data/lists/test_Mic_5.list
$BASEDIR/data/lists/test_Mic_6.list)
test_transcChannels=($BASEDIR/data/transcriptions/test_CH_1.txt
$BASEDIR/data/transcriptions/test_CH_3.txt
$BASEDIR/data/transcriptions/test_CH_4.txt
$BASEDIR/data/transcriptions/test_CH_5.txt
$BASEDIR/data/transcriptions/test_CH_6.txt)
# ----- Number of microphones and ASR systems
MICROPHONES=5
ASR_SYSTEMS=1
folds=5 # K-fold cross validation
# ----- QE Algorithm
QE=RR
RR_Iter=100
MLR_Tune=Yes
# ----- The type of Features to be used
F_SIG=0 # use signal based features otherwise set it to 0
F_LM=1 # use word-level language model features otherwise set it to 0
F_POS=1 # use word-level part-of-speech features otherwise set it to 0
F_LEX=1 # use word-level lexical features otherwise set it to 0

```

Figure 6: The configuration file.

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.2015] Shahab Jalalvand, Matteo Negri, Daniele Falavigna, and Marco Turchi. 2015. Driving ROVER With Segment-based ASR Quality Esti-

- mation. 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.