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 to perform 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 textitHow 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 and Daniele Falavigna from FBK (Fondazione Bruno Kessler) research center.

1.1 Applications

Any task that uses ASR output can take advantage of this toolkit. Spoken translation, multiple distant ASR systems and ASR system combination (such as ROVER) are some of the downstream tasks. To show the effectiveness of ASR QE, we provide some experiments in the toolkit for performing ROVER system combination with ranked inputs defined by ASR QE. The full set of results and experiments can be found in [ACL2015].

2 User's Guide

2.1 System requirements

TranscRater is implemented using a set of bash scripts and python codes. Up to the time of writing this manual it is only runnable under Linux. Soon, we will update the toolkit with the possibility of being installed under Windows.
On a Linux system, you will require to install the following items:

- The Java and python required:
- Java 8 (JDK-1.8)
- Python 2.7.6 (or above - only 2.7 stable distributions)
- NumPy and SciPy (NumPy $>=$ 1.6.1 and SciPy $>=$ 0.9)
- scikit-learn (version 0.15.2)

The tools and libraries that are needed to perform feature extraction are:

- ”OpenSmile” for extracting signal-based feature
- ”RNNLM” for extracting the RNNLM probabilities
- ”SRILM” for extracting the SRILM probabilities
- ”TreeTagger” for extracting the part-of-speech tags and scores
- “RankLib” for training and test the Machine-Learned Ranking (MLR) methods.

The paths for the required tools and libraries are to be set in “configuration.conf” file under ”conf” folder.

“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 function named “SMILExtract” to extract the signal features, therefore the user is asked to put the path to this function into the “./conf/configuration.conf” file after installing the library.

“RNNLM” is a toolkit to train and test 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” function to calculate the probability of the words. The user is asked to set the path to this function in the “./conf/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 function named “ngram” so that the user is asked to set the path to this function in the “./conf/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 function named “tree-tagger”, so that the user is asked to set the path to this function in “./conf/configuration.conf”.

“RankLib” is a library for machine-learned ranking (or learning to rank) algorithms. It can be freely downloaded from
`https://people.cs.umass.edu/~vdang/data/RankLib-v2.1.tar.gz`
Once you download and extract the folder, you must put the path to the “RankLib.jar” file in the “./conf/configuration.conf” file. In this toolkit we only use RandomForest 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

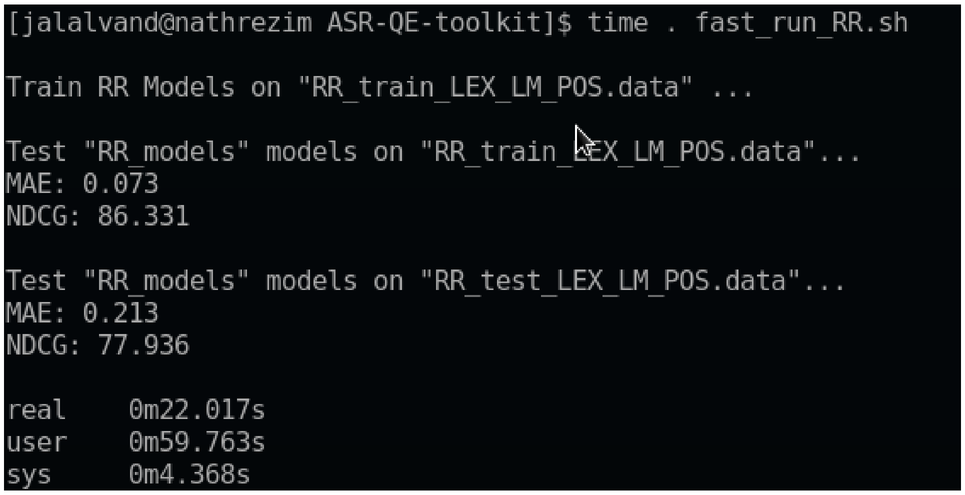
Once you have all the aforementioned requirements then you can start using TrancRater.

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 to run the commands. In the root folder run the following command:

```
. fast_run_RR.sh
```

This trains an XRT 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 with 8 Gig RAM. The output looks like Figure 1.



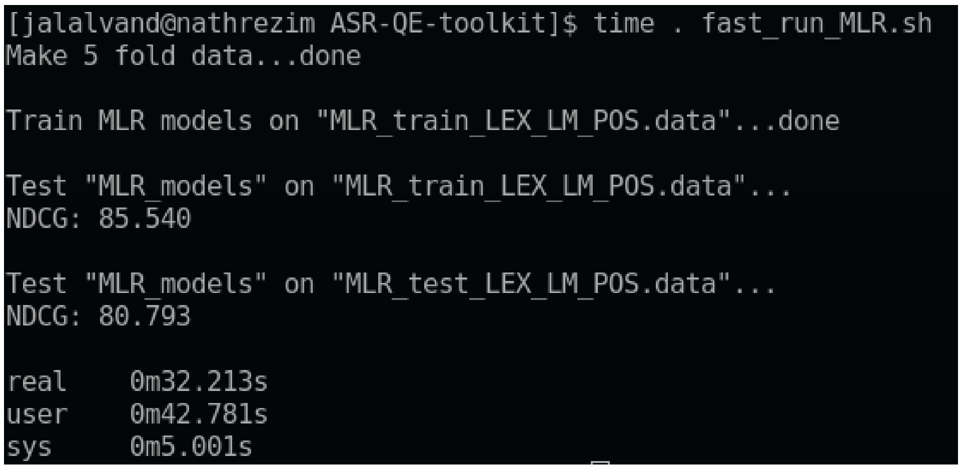
```
[jalalvand@nathrezim ASR-QE-toolkit]$ time . fast_run_RR.sh

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.



```
[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.

2.2.2 Configuration file

The user needs to define the following information in the configuration file.

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

OPENSMILEDIR = 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 training auto transcriptions

MICROPHONES* = the number of microphones
```

```
ASR_SYSTEMS*= the number of ASR systems

folds* = the number of folds

QE* = "RR" for regression methods; "MLR" for ranking
RR_Iter = the number of training iterations for RR methods
MLR_Tune = "Yes" if you want to optimize the parameters on training

# at least one feature type 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 features
F_POS=0 set "1" if you want to use part-of-speech based features
```

Note that the stared fields must be set in the configuration file. Also the toolkit makes some temporary folders in the base directory to store the temporary files and the results.

2.2.3 Input files

The format of the audio files is as: The format of the wave files is:
`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-light[1] 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 paper.

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 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.

[1] `http://svmlight.joachims.org/`

```
     ---RR file format Single Channel-----------
     0.429 qid:01 1:5.01 2:9.58 3:-4.49 … #CH_1
a)   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
     -------------------------------------------
     ---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
c)   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 3: The format of the internal data files.

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.

3.1.1 Label computation

When the reference file is in ”train.ref” and the automatic transcription is in train.hyp, then the WER of each transcription is computed using the following function:
`./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 and also it will be used to derive the ranks as the labels for the ranking machines.

3.1.2 feature vectores

Given the assumption that the transcription files are defined in the configuration file in ”train_transcChannels” variable and the auido files are defined in ”train_wavChannels”, then the following commands extracts respectively the signal, lexical, language model and POS features:

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

and they store the resulting files in:

```
./data/features/train_CH_1_LEX.feat
./data/features/train_CH_1_LM.feat
./data/features/train_CH_1_POS.feat
```

The same for the other transcription channels (CH_2, CH_3, etc.) if they exist.

3.2 Machine Learning

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

3.2.1 Regression (RR)

The following command will prepare the data file for training the 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.
When "RR_train_FEATURE.data" file is ready then one can train a regression model using the following command and store it in "RR_models" folder:

```
./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 Ranking by Regression

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 prepare the data based on the feature types and the ranks as the labels:

```
./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.data k MLR_models
```

The same as RR approach, to test the MLR models we need to prepare the test data by defining the types of Features and the channels:

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

And finally by the following command we can predict the rankings:

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