Automatic speech recognition experiments
Additional information

This short document provide some additional information about the automatic speech recognition experiments to be carried out on the digits data using the pocketsphinx tools under python.

Table of contents

1 About the speech recognition experiments

1.1 Decoding sequences of digits with a jsgf grammar

Start from one program example, either 'ps_exemples/decoder_jsgf.py' or 'ps_exemples/decoder_utt_jsgf.py'.

The difference between the two programs lies in the way the speech data is sent to the decoder. The first program sends the speech data to the decoder in small packs (small data buffer), whereas the second one sends all the speech data of a given utterance at once. As in these speech recognition experiments you are dealing with speech files of a rather small size (sequences of a few digits), both approaches can be used.

WARNING. In these program examples, the decoding is done twice for each speech file: a first time using an ngram grammar (corresponding to the turtle application), and a second time using the jsgf grammar of the turtle application. When you will adapt the program to process sequences of digits with a jsgf grammar, you can / should keep only the jsgf based decoding (using the lexicon and the grammar you have developed for digit sequences).

Suggestion:

- Start from a program example, and adapt it to process a speech file containing a sequence of digits.
    - Check that the compilation of the grammar you have developed works correctly
    - Check that the decoding is OK
    - Check that the output (recognition hypothesis) corresponds to the grammar you are using. For example, if you are decoding with a grammar specifying 3-digit sequences, the recognition hypothesis should have exactly three digits.
- Once the processing of a file is OK, modify the program in order to process a bunch of digit files, and write the results (speech recognition hypothesis) in a text file.
  Use one line for each processed utterance:
  <path_and_name_of_the_speech_file > <recognized_words_for_this_utterance>

1.2 Evaluating the word error rate

See also the corresponding slides in the presentation of the speech recognition experiments

Using the result text files corresponding to your experiments (language model, lexicon, and selected speech data subsets), create the 'data.ref' and 'data.hyp' files.

As mentioned in the slides, each line corresponds to a sequence of words (in your case a sequence of digits).

- A line of the 'data.ref' file corresponds to a sequence of pronounced words (i.e. reference).
- A line of the 'data.hyp' file corresponds to a sequence of recognized words.

WARNING. The two files must have exactly the same number of lines. The n-th line of the file 'data.ref' must be synchronized with the n-th line of the file 'data.hyp' ; i.e., reference (in 'data.ref') and recognized words (in 'data.hyp') corresponding to the same speech file.

Once the two files are ready, you can run the program 'wer.py' to evaluate the speech recognition performance.

Do not forget to use the option « -i » to obtain the list of the alignments between the references and the recognized words.

By looking at these alignments, you will easily notice the various types of errors: insertions, deletions, and substitutions.

Note that, for each utterance, the associated reference (i.e. the list of digits pronounced for that utterance) is available next to the speech file. For each utterance, you will get the name of the reference file by replacing the extension '.raw' by '.ref'.

2 A few remarks

2.1 Raw files

A 'raw' speech file contains only the values of the speech waveform samples.

Unlike the 'wav' format, the 'raw' file has no header, so there is no indication of the speech sampling characteristics: sampling rate, size of sample numeric values, …

Pocketsphinx process speech files in 'raw' format, thus the speech files are given in this format.

For information, the characteristics of the 'raw' speech files are the following:

- 16000 Hz sampling rate
- 16 bits signed integer values

2.2 Compilation of the jsgf grammar

When pocketsphinx compile the jsgf grammar, it provides no information on the type of errors, if any.

So if the grammar does not compile, check for frequent errors such as:

- Incomplete or erroneous specification in the jsgf file. For example:
    - Usage of nonexistent keywords
    - Bad syntax
    - Usage of undefined rules
    - …
- Mismatch between the words specified in the grammar and those defined in the specified lexicon. For example, the grammar refer to a word that is not present in the lexicon.

2.3 Decoding error

During decoding, in few cases, it may happened that pocketsphinx does not find a result (speech recognition hypothesis) when processing a speech file. This lead to an error message of the following type:

```
ERROR: "fsg_search.c", line 940: Final result does not match the grammar in frame 66
Traceback (most recent call last):
  File "toto.py", line 61, in <module>
    print ('%s :: %s [score=%d, conf=%.2f]' % (audiofn, hypothesis.hypstr,
hypothesis.best_score, logmath.exp(hypothesis.prob)))
AttributeError: 'NoneType' object has no attribute 'hypstr'
```

In such case, a solution consists in checking that the result actually exists before writing it, for example:

```
    hypothesis = decoder.hyp()

        if hypothesis.hypstr != None:
```

then, you can:

- Either ignore the speech file leading to such error (i.e. writing nothing in the output file)
- Or writing an empty string as recognition result for this file (this will thus be counted as deletion errors)