sphinxwrapper Documentation

Release 1.2.0

Dane Finlay

Dec 31, 2022

Contents:

Release v1.2.0

Simplified Python API for recognising speech with CMU Pocket Sphinx

CHAPTER 1

Introduction to sphinxwrapper

Simplified Python API for the CMU Pocket Sphinx speech recogniser

This package provides a simple API for recognising speech using CMU Pocket Sphinx, an open source, lightweight speech recognition engine. More information on CMU Pocket Sphinx, and other CMU speech recognition libraries, may be found at cmusphinx.github.io.

There are some usage examples in the repository's examples folder demonstrating how to use this library to scan and process speech audio from a microphone. Each of these examples require the PyAudio package, which may be installed by running the following command:

```
pip install pyaudio
```

1.1 Installation & dependencies

To install this package via pip, run the following command:

```
pip install sphinxwrapper
```

If you are installing in order to *develop* sphinxwrapper, clone/download the repository, move to the root directory and run:

```
pip install -e .
```

Either of the above commands will also install version 0.1.15 of the required pocketsphinx-python package.

1.2 Usage example

The following example demonstrates how to use `sphinxwrapper` and PyAudio to scan and interpret audio from the microphone using the default language model and dictionary.

```
import os
import time

from pyaudio import PyAudio, paInt16

from sphinxwrapper import PocketSphinx, DefaultConfig

# Initialise a Pocket Sphinx decoder with the default configuration.
config = DefaultConfig()
config.set_string("-logfn", os.devnull)  # Suppress log output.
ps = PocketSphinx(config)

# Define decoder callback functions.
def speech_start_callback():
    print("Speech started.")

def hypothesis_callback(hyp):
    hypstr = hyp.hypstr if hyp else None
    print("Hypothesis: %r" % hypstr)

# Set decoder callback functions.
ps.speech_start_callback = speech_start_callback
ps.hypothesis_callback = hypothesis_callback

# Open an audio stream on the default input audio device.
p = PyAudio()
stream = p.open(format=paInt16, channels=1, rate=16000, input=True,
                frames_per_buffer=2048, input_device_index=None)
stream.start_stream()

# Recognise from the microphone in a loop until interrupted.
try:
    print("Listening... Press Ctrl+C to exit...")
    while True:
        ps.process_audio(stream.read(2048))
        time.sleep(0.1)
except KeyboardInterrupt:
    stream.stop_stream()
    p.terminate()
```

1.3 Python versions

This package has been written for Python 2.7 and above. It should work the same way for each supported version. Please file an issue if you encounter a problem specific to the Python version you're using.

1.4 Documentation

The documentation for this project is written in reStructuredText and built using the Sphinx documentation engine.

Run the following commands in the repository folder to build it locally:

```
cd docs
pip install -r requirements.txt
make html
```

CHAPTER 2

sphinxwrapper Python package

This section documents the available classes, methods, properties and functions.

2.1 CMU Pocket Sphinx Decoder Class

class sphinxwrapper.pocketsphinx_wrap.**PocketSphinx**(*config=None*)

Pocket Sphinx decoder subclass with processing methods providing callback functionality, as well as some other things.

This class will try to set the '-hmm' and '-dict' configuration arguments automatically if they are not set prior to initialisation.

If no search argument is present, the class will also try to set the '-lm' argument so that the default language model is used. Search arguments include the following:

- '-lm'
- '-jsgf'
- '-fsg'
- '-keyphrase'
- '-kws'

Construct arguments:

- *config* – Decoder configuration object. Will be initialised using `default_config()` if unspecified.

Note: An error will be raised if the configuration object specifies more than search argument. In this event, the decoder class will not be initialised.

---

active_search

The name of the currently active Pocket Sphinx search.

If the setter is passed a name with no matching Pocket Sphinx search, a `RuntimeError` will be raised.

Return type str

batch_process(*buffers*, *no_search=False*, *full_utterance=False*, *use_callbacks=True*)
Process a list of audio buffers and return the speech hypothesis, if there one.

This method uses the *process_audio()* method.

Note: If *buffers* contains more than one utterance worth of audio, only the final `Hypothesis` object is returned.

Parameters

- **buffers** (*list*) – List of audio buffers
- **no_search** (*bool*) – Whether to perform feature extraction, but no recognition yet (default: *False*).
- **full_utterance** (*bool*) – Whether this block of data contains a full utterance worth of data (default: *False*). This may produce more accurate results.
- **use_callbacks** (*bool*) – Whether speech start and hypothesis callbacks should be called (default: *True*).

Return type Hypothesis | None

Returns The decoder's hypothesis, or *None* if there isn't one (yet).

end_utt()
Ends the current utterance if one was in progress.

Does nothing if no utterance is in progress.

get_in_speech()
Check if the last audio buffer contained speech.

Returns Whether the last audio buffer contained speech.

Return type bool

hypothesis_callback
Function invoked when the decoder has finished processing speech.

To use this callback, set it to a callable that takes one positional argument, the decoder's hypothesis:

```
ps = PocketSphinx()

def callback(hyp):
    print(hyp)

ps.hypothesis_callback = callback
```

To disable this callback, set it to `None` (default).

process_audio(*buf*, *no_search=False*, *full_utterance=False*, *use_callbacks=True*)
Process an audio buffer and return the speech hypothesis, if there is one.

This method processes the given buffer with the `process_raw()` decoder method, invoking *speech_start_callback* and *hypothesis_callback* when appropriate.

Parameters

- **buf** (*str*) – Audio buffer

- **no_search** (*bool*) – Whether to perform feature extraction, but no recognition yet (default: *False*).
- **full_utterance** (*bool*) – Whether this block of data contains a full utterance worth of data (default: *False*). This may produce more accurate results.
- **use_callbacks** (*bool*) – Whether speech start and hypothesis callbacks should be called (default: *True*).

Return type Hypothesis | None

Returns The decoder's hypothesis, or *None* if there isn't one (yet).

set_kws_list(*name*, *kws_list*)
Set a keyword-list search which, when active, scans input audio for keywords defined in the specified list or dictionary.

Parameters

- **name** (*str*) – Search name
- **kws_list** (*list | dict*) – Dictionary of words to threshold value. Can also be a list of 2-tuples.

speech_start_callback
Function invoked when speech is first detected.

To use this callback, set it to a callable that takes no arguments:

```
ps = PocketSphinx()

def callback():
    print("Speech started.")

ps.speech_start_callback = callback
```

To disable this callback, set it to `None` (default).

start_utt()
Starts a new utterance if one is not already in progress.

Does nothing if an utterance is already in progress.

utt_ended
Whether there is no utterance in progress.

Return type bool

utt_idle
Whether an utterance is in progress, but no speech has been detected yet.

Return type bool

utt_started
Whether an utterance is in progress and speech has been detected.

Return type bool

2.2 Decoder Configuration

exception sphinxwrapper.config.**ConfigError**
Error raised if something is wrong with the decoder configuration.

sphinxwrapper.config.**dict_file_extensions = ['.dic', '.dict']**
List of acceptable file extensions for pronunciation dictionary files.

sphinxwrapper.config.**hmm_dir_files = ['feat.params', 'mdef', 'noisedict', 'transition_matr**
List of files that must be present in a hidden Markov model directory.

sphinxwrapper.config.**lm_file_extensions = ['.lm', '.lm.bin']**
List of acceptable file extensions for language model files.

sphinxwrapper.config.**search_arguments_set**(*config*)
Get a list of search arguments set for a given `Config` object.

Search arguments include:

- '-lm' (default)
- '-fsg'
- '-jsgf'
- '-keyphrase'
- '-kws'

Parameters **config** (*Config*) – Decoder configuration object

Returns list

sphinxwrapper.config.**set_hmm_and_dict_paths**(*config*, *model_path=None*)
This function will try to find a hidden Markov model (HMM) directory and a pronunciation dictionary file under *model_path*. If found, the function sets the '-hmm' and '-dict' arguments in the given `Config` object appropriately and returns `True`. If an argument is already set, its value is not changed. If both arguments were already set, the function returns `False`.

By default, this function requires an HMM directory to contain each of the following files:

- *feat.params*
- *mdef*
- *noisedict*
- *transition_matrices*
- *variances*

Likewise, the pronunciation dictionary file must, by default, have the file extension *.dic* or *.dict* in order for this function to use it.

An error will be raised if the function finishes without both arguments set in the `Config` object.

Parameters

- **config** (*Config*) – Decoder configuration object
- **model_path** (*str*) – Path to search for the HMM directory and pronunciation dictionary file. The Pocket Sphinx `get_model_path()` function is used if this parameter is unspecified.

Raises ConfigError

Returns Whether any configuration argument was changed.

Return type bool

sphinxwrapper.config.**set_lm_path**(*config*, *model_path=None*)

This function will try to find the language model file under *model_path*, set the '-lm' argument for the given `Config` object and return `True`. If the argument is already set, its value is not changed and the function returns `False`.

By default, only files ending with *.lm* or *.lm.bin* will be considered.

An error will be raised if the path was not found.

Parameters

- **config** (*Config*) – Decoder configuration object
- **model_path** (*str*) – Path to search for the language model file. The Pocket Sphinx `get_model_path()` function is used if this parameter is unspecified.

Raises ConfigError

Returns Whether any configuration argument was changed.

Return type bool

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index