sphinxwrapper Documentation

Release 1.2.0

Dane Finlay

Jun 22, 2019

Contents:

Release v1.2.0

Alternative Python API for recognising speech with CMU Pocket Sphinx

CHAPTER 1

Introduction to sphinxwrapper

Alternative Python API for recognising speech with CMU Pocket Sphinx

This package aims to provide a simple API for recognising speech using the Pocket Sphinx API. Pocket Sphinx is an open source, lightweight speech recognition engine. You can read more about the CMU Sphinx speech recognition projects here.

There are some usage examples in the repository's examples folder.

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 the required pocketsphinx-python package.

The usage examples for `sphinxwrapper` require the cross-platform pyaudio Python package. It can be installed by running the following:

```
pip install pyaudio
```

1.2 Usage example

The following is a simple usage example showing how to use the `sphinxwrapper` package to make Pocket Sphinx continuously recognise speech from the microphone using the default decoder configuration.

```
import time

from pyaudio import PyAudio, paInt16

from sphinxwrapper import PocketSphinx

# Set up a Pocket Sphinx decoder with the default config
ps = PocketSphinx()

# Set up and register a callback function to print Pocket Sphinx's
# hypothesis for  recognised speech
def print_hypothesis(hyp):
    # Get the hypothesis string from the Hypothesis object or None, then
    # print it
    speech = hyp.hypstr if hyp else None
    print("Hypothesis: %s" % speech)

ps.hypothesis_callback = print_hypothesis

# Decode from the default audio input device continously.
p = PyAudio()
stream = p.open(format=paInt16, channels=1, rate=16000, input=True,
                frames_per_buffer=2048)
while True:
    ps.process_audio(stream.read(2048))
    time.sleep(0.1)
```

1.3 Python versions

This package has been written for Python 2.7 and above. It should work exactly the same way for each supported version. please file an issue if you come across any problems that are specific to the Python version you're using.

1.4 Future CMU Sphinx API changes

As the CMU Sphinx libraries are pre-alpha, there may be future changes that break this package in some way. I'm happy to fix any such issues, just file an issue.

1.5 Documentation

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

Run the following 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 Classes

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

Pocket Sphinx decoder subclass with processing methods providing callback functionality and other things.

This class will try to set required config options, such as '-hmm', '-dict' and/or '-lm', in the config object automatically if they are not set.

Construct arguments:

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

Note: The decoder class will not be initialised if the configuration object specifies more than search argument.

---

active_search

The name of the currently active Pocket Sphinx search.

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

Returns str

batch_process(*buffers*, *no_search=False*, *full_utterance=False*, *use_callbacks=True*)

Process a list of audio buffers and return the speech hypothesis or use the decoder callbacks if use_callbacks is True.

end_utt()

Ends the current utterance if one was in progress.

This method is useful for resetting processing of audio via the *process_audio()* method.

It will *not* raise an error if no utterance was in progress.

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

This method will also move utterance state from idle to started.

Returns whether the last audio buffer contained speech.

Return type bool

hypothesis_callback
Callback called with Pocket Sphinx's hypothesis for what was said.

process_audio(*buf*, *no_search=False*, *full_utterance=False*, *use_callbacks=True*)
Process audio from an audio buffer using the `process_raw()` decoder method.

The speech start and hypothesis callbacks will be called if and when necessary.

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*).

set_kws_list(*name*, *kws_list*)
Set a keywords Pocket Sphinx search with the specified name taking a keywords list as a Python dictionary.

This method generates a temporary keywords list file and calls the `set_kws()` decoder method with its path.

Parameters

- **name** (*str*) – search name
- **kws_list** – dictionary of words to threshold value. Can also be a list of 2-tuples.

speech_start_callback
Callback for when speech starts.

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

This method will *not* raise an error if an utterance is in progress (started or idle) already.

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.

get_in_speech() would return `False` if this returns `True`.

Return type bool

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

get_in_speech() would return `True` if this returns `True`.

Return type bool

2.2 Decoder Configuration

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

sphinxwrapper.config.**search_arguments_set**(*config*)
This function returns the 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 the HMM directory and dictionary file paths in *model_path* and set the '-hmm' and '-dict' arguments in the given Config object.

An error will be raised if any of the paths were not found.

Parameters

- **config** (*Config*) – decoder configuration object
- **model_path** (*str*) – path to search for HMM and dictionary. The Pocket Sphinx get_model_path() function is used if the parameter is unspecified.

Raises ConfigError

sphinxwrapper.config.**set_lm_path**(*config*, *model_path=None*)
This function will try to find the LM file in model_path and set the '-lm' argument for the given Config object.

Only files ending with *.lm* or *.lm.bin* will be considered. An error will be raised if an LM file cannot be found.

Parameters

- **config** (*Config*) – decoder configuration object
- **model_path** (*str*) – path to search for the LM file. The Pocket Sphinx get_model_path() function is used if the parameter is unspecified.

Raises ConfigError

CHAPTER 3

Old C Extension module

This project used to be in the form of a Python module written in C (i.e. a Python C extension). The reason for this is that the CMU Pocket Sphinx and sphinxbase libraries are both written in C and I developed *sphinxwrapper* to abstract some of functionality provided by the pocketsphinx_continuous program so I could use it in my Pocket Sphinx dragonfly engine.

The *sphinxwrapper* package now interacts with CMU Pocket Sphinx via the pocketsphinx-python package instead, so compiling isn't necessary any more.

The C extension version still exists in the repository's *extension* folder for historical reasons. It is **not recommended** that you use this version for various reasons.

The API is similar to the current version with some differences:

- It uses the platform's *sphinxbase* audio device implementation for reading data from the microphone and cannot be used with *pyaudio* without some hacking.
- Some functions and properties behave differently or just don't exist.
- Most of the classes, functions and methods provided by the pocketsphinx-python package are not present.

Writing the C extension was mostly a learning experience for me. Python modules written in C are dreadfully difficult to write and maintain, plus the CMU Sphinx libraries used already have very good Python compatibility through Swig.

Like the current version of *sphinxwrapper*, the C extension should work correctly with Python versions 2.7.x and 3.x.

I might update it at some point to use the same API as the current version.

3.1 Compiling & Installing

To compile this version of *sphinxwrapper*, you need to have the following dependencies installed:

- sphinxbase
- pocketsphinx
- Python C headers

You can install the dependencies using the following commands on Debian GNU/Linux. They might also work on Debian-derived systems such as Ubuntu and will be similar on other systems.

I've tested this with *pocketsphinx* and *sphinxbase* versions **0.8+5prealpha-3**.

```
# Install the required library headers
sudo apt install libpocketsphinx-dev libpocketsphinx3 libsphinxbase-dev libsphinxbase3

# Install the Python headers for your version of Python
sudo apt install python-dev
sudo apt install python3-dev
```

You can compile and install the required CMU Sphinx libraries from source instead. Have a look at the project for information

You probably also want to install the `pocketsphinx-en-us` package for the default models and dictionary, unless you specify the *-hmm*, *-dict* and other arguments (e.g. *-lm*) yourself.

You will also need the C/C++ compiler that your version of Python was compiled with. Normally this is GCC on Linux or Clang on Mac OS/*BSD. You can identify the required compiler by starting an interpreter. For example, my Debian 9 machine has Python 2.7.13 comiled with GCC v6.3.0:

```
Python 2.7.13 (default, Sep 26 2018, 18:42:22)
[GCC 6.3.0 20170516] on linux2
```

You should be able to compile and install it by running the following in the repository's *extension* folder:

```
cd extension
python setup.py install
```

I will probably not distribute this version of *sphinxwrapper* on PyPI.

3.1.1 Microsoft Windows

I would not recommend using this version of *sphinxwrapper* on Windows because compiling Python C/C++ extensions on that platform is not easy. It should work using `python setup.py install`, but you'll probably need to download and install a few things first. The Python wiki page on Windows Compilers should help.

Please note that you will likely have no luck getting this working on *Cygwin* because *libsphinxbasead* won't compile in that environment, at least it didn't the last time I tried. You're better off using a virtual machine with a Linux distribution if you're really determined to use this on Windows.

3.2 Usage example

As mentioned above, there are a few differences with this version of the package. The below usage example should work. It is similar to the pocketsphinx_continuous program.

```
import os
import time

from sphinxwrapper import PocketSphinx, AudioDevice


def speech_start_callback():
    print("Speech started.")
```

(continues on next page)

(continued from previous page)

```
def hyp_callback(s):
    print("Hypothesis: %s" % s)


# Initialise a decoder with the default configuration.
# ps = PocketSphinx()

# The decoder optionally accepts an command-line argument list for
# decoder configuration. The following will suppress log output.
ps = PocketSphinx(["-logfn", os.devnull])

# Set up callback functions.
ps.speech_start_callback = speech_start_callback
ps.hypothesis_callback = hyp_callback

# Recognise from the mic in a loop.
ad = AudioDevice()
ad.open()
ad.record()
while True:
    audio = ad.read_audio()
    ps.process_audio(audio)
    time.sleep(0.1)
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index