# Python Package Index: Speech Recognition 3.8.1

转录至：https://www.jianshu.com/p/0cc915a28de3

## ▌语言识别工作原理概述

语音识别源于 20 世纪 50 年代早期在贝尔实验室所做的研究。早期语音识别系统仅能识别单个讲话者以及只有约十几个单词的词汇量。现代语音识别系统已经取得了很大进步，可以识别多个讲话者，并且拥有识别多种语言的庞大词汇表。

语音识别的首要部分当然是语音 ‹通过麦克风 ›语音便从物理声音被转换为电信号，然后通过模数转换器转换为数据。一旦被数字化，就可适用若干种模型，将音频转录为文本。

大多数现代语音识别系统都依赖于隐马尔可夫模型（HMM）。其工作原理为：语音信号在非常短的时间尺度上（比如 10 毫秒）可被近似为静止过程，即一个其统计特性不随时间变化的过程。

许多现代语音识别系统会在 HMM 识别之前使用神经网络，通过特征变换和降维的技术来简化语音信号。也可以使用语音活动检测器（VAD）将音频信号减少到可能仅包含语音的部分。

幸运的是，对于 Python 使用者而言，一些语音识别服务可通过 API 在线使用，且其中大部分也提供了 Python SDK。

## ▌选择 Python 语音识别包

PyPI 中有一些现成的语音识别软件包。其中包括：

•apiai

•google-cloud-speech

•pocketsphinx

•SpeechRcognition

•watson-developer-cloud

•wit

一些软件包（如 wit 和 apiai ）提供了一些超出基本语音识别的内置功能，如识别讲话者意图的自然语言处理功能。其他软件包，如谷歌云语音，则专注于语音向文本的转换。

其中，SpeechRecognition 就因便于使用脱颖而出。

识别语音需要输入音频 ,而在 SpeechRecognition 中检索音频输入是非常简单的，它无需构建访问麦克风和从头开始处理音频文件的脚本，只需几分钟即可自动完成检索并运行。

SpeechRecognition 库可满足几种主流语音 API ，因此灵活性极高。其中 Google Web Speech API 支持硬编码到 SpeechRecognition 库中的默认 API 密钥，无需注册就可使用。SpeechRecognition 以其灵活性和易用性成为编写 Python 程序的最佳选择。

## ▌安装 SpeechRecognation

SpeechRecognition 兼容 Python2.6 , 2.7 和 3.3+，但若在 Python 2 中使用还需要一些额外的安装步骤。本教程中所有开发版本默认 Python 3.3+。

读者可使用 pip 命令从终端安装 SpeechRecognition：

$ pip install SpeechRecognition

安装完成后请打开解释器窗口并输入以下内容来验证安装：

>>> importspeech_recognitionassr

>>> sr.__version__

'3.8.1'

注：不要关闭此会话，在后几个步骤中你将要使用它。

若处理现有的音频文件，只需直接调用 SpeechRecognition ，注意具体的用例的一些依赖关系。同时注意，安装 PyAudio 包来获取麦克风输入。

## ▌识别器类

SpeechRecognition 的核心就是识别器类。

Recognizer API 主要目是识别语音，每个 API 都有多种设置和功能来识别音频源的语音，分别是：

recognize_bing(): Microsoft Bing Speech

recognize_google(): Google Web Speech API

recognize_google_cloud(): Google Cloud Speech – requires installation of the google-cloud-speech package

recognize_houndify(): Houndify by SoundHound

recognize_ibm(): IBM Speech to Text

recognize_sphinx(): CMU Sphinx – requires installing PocketSphinx

recognize_wit(): Wit.ai

以上七个中只有 recognition_sphinx（）可与 CMU Sphinx 引擎脱机工作， 其他六个都需要连接互联网。

SpeechRecognition 附带 Google Web Speech API 的默认 API 密钥，可直接使用它。其他六个 API 都需要使用 API 密钥或用户名/密码组合进行身份验证，因此本文使用了 Web Speech API。

现在开始着手实践，在解释器会话中调用 recognise_google（）函数。

>>> r.recognize_google()

屏幕会出现：

Traceback (most recentcalllast):

File"", line1,in

TypeError: recognize_google()missing1requiredpositional argument:'audio_data'

相信你已经猜到了结果，怎么可能从空文件中识别出数据呢？

这 7 个 recognize_*()识别器类都需要输入 audio_data 参数，且每种识别器的 audio_data 都必须是 SpeechRecognition 的 AudioData 类的实例。

AudioData 实例的创建有两种路径：音频文件或由麦克风录制的音频，先从比较容易上手的音频文件开始。

## ▌音频文件的使用

首先需要下载音频文件（https://github.com/realpython/python-speech-recognition/tree/master/audio_files），保存到 Python 解释器会话所在的目录中。

AudioFile 类可以通过音频文件的路径进行初始化，并提供用于读取和处理文件内容的上下文管理器界面。

**支持文件类型**

SpeechRecognition 目前支持的文件类型有：

WAV: 必须是 PCM/LPCM 格式

AIFF

AIFF-C

FLAC: 必须是初始 FLAC 格式；OGG-FLAC 格式不可用

若是使用 Linux 系统下的 x-86 ，macOS 或者是 Windows 系统，需要支持 FLAC 文件。若在其它系统下运行，需要安装 FLAC 编码器并确保可以访问 flac 命令。

**使用 record() 从文件中获取数据**

在解释器会话框键入以下命令来处理“harvard.wav”文件的内容：

```
>>> harvard = sr.AudioFile('harvard.wav')

>>> withharvardassource:

... audio = r.record(source)
```

...

通过上下文管理器打开文件并读取文件内容，并将数据存储在 AudioFile 实例中，然后通过 record（）将整个文件中的数据记录到 AudioData 实例中，可通过检查音频类型来确认：

>>> type(audio)

现在可以调用 recognition_google（）来尝试识别音频中的语音。

>>> r.recognize_google(audio)

'the stale smell of old beer lingers it takes heat

to bring out the odor a cold dip restores health and

zest a salt pickle taste fine with ham tacos al

Pastore are my favorite a zestful food is the hot

cross bun'

以上就完成了第一个音频文件的录制。

**利用偏移量和持续时间获取音频片段**

若只想捕捉文件中部分演讲内容该怎么办？record() 命令中有一个 duration 关键字参数，可使得该命令在指定的秒数后停止记录。

例如，以下内容仅获取文件前四秒内的语音：

>>> withharvardassource:

... audio = r.record(source, duration=4)

...

>>> r.recognize_google(audio)

'the stale smell of old beer lingers'

在 with 块中调用 record()命令时，文件流会向前移动。这意味着若先录制四秒钟，再录制四秒钟，则第一个四秒后将返回第二个四秒钟的音频。

>>> withharvardassource:

... audio1 = r.record(source, duration=4)

... audio2 = r.record(source, duration=4)

...

>>> r.recognize_google(audio1)

'the stale smell of old beer lingers'

>>> r.recognize_google(audio2)

'it takes heat to bring out the odor a cold dip'

除了指定记录持续时间之外，还可以使用 offset 参数为 record() 命令指定起点，其值表示在开始记录的时间。如：仅获取文件中的第二个短语，可设置 4 秒的偏移量并记录 3 秒的持续时间。

>>> withharvardassource:

... audio = r.record(source, offset=4, duration=3)

...

>>> recognizer.recognize_google(audio)

'it takes heat to bring out the odor'

在事先知道文件中语音结构的情况下，offset 和 duration 关键字参数对于分割音频文件非常有用。但使用不准确会导致转录不佳。

>>> withharvardassource:

... audio = r.record(source, offset=4.7, duration=2.8)

...

>>> recognizer.recognize_google(audio)

'Mesquite to bring out the odor Aiko'

本程序从第 4.7 秒开始记录，从而使得词组“it takes heat to bring out the odor”，中的“it t”没有被记录下来，此时 API 只得到“akes heat”这个输入，而与之匹配的是“Mesquite”这个结果。

同样的，在获取录音结尾词组“a cold dip restores health and zest”时 API 仅仅捕获了“a co”，从而被错误匹配为“Aiko”。

噪音也是影响翻译准确度的一大元凶。上面的例子中由于音频文件干净从而运行良好，但在现实中，除非事先对音频文件进行处理，否则不可能得到无噪声音频。

**噪声对语音识别的影响**

噪声在现实世界中确实存在，所有录音都有一定程度的噪声，而未经处理的噪音可能会破坏语音识别应用程序的准确性。

要了解噪声如何影响语音识别，请下载“jackhammer.wav”（https://github.com/realpython/python-speech-recognition/tree/master/audio_files) 文件，并确保将其保存到解释器会话的工作目录中。文件中短语“the stale smell of old beer lingers”在是很大钻墙声的背景音中被念出来。

尝试转录此文件时会发生什么？

```
>>> jackhammer = sr.AudioFile('jackhammer.wav')

>>> withjackhammerassource:

... audio = r.record(source)

...

>>> r.recognize_google(audio)

'the snail smell of old gear vendors'
```

那么该如何处理这个问题呢？可以尝试调用 Recognizer 类的 adjust_for_ambient_noise（）命令。

```
>>> withjackhammerassource:

... r.adjust_for_ambient_noise(source)

... audio = r.record(source)
```

...

>>> r.recognize_google(audio)

'still smell of old beer vendors'

这样就与准确结果接近多了，但精确度依然存在问题，而且词组开头的“the”被丢失了，这是什么原因呢？

因为使用 adjust_for_ambient_noise（）命令时，默认将文件流的第一秒识别为音频的噪声级别，因此在使用 record（）获取数据前，文件的第一秒已经被消耗了。

可使用 duration 关键字参数来调整 adjust_for_ambient_noise（）命令的时间分析范围，该参数单位为秒，默认为 1，现将此值降低到 0.5。

>>> withjackhammerassource:

... r.adjust_for_ambient_noise(source, duration=0.5)

... audio = r.record(source)

...

>>> r.recognize_google(audio)

'the snail smell like old Beer Mongers'

现在我们就得到了这句话的“the”，但现在出现了一些新的问题——有时因为信号太吵，无法消除噪音的影响。

若经常遇到这些问题，则需要对音频进行一些预处理。可以通过音频编辑软件，或将滤镜应用于文件的 Python 包（例如 SciPy）中来进行该预处理。处理嘈杂的文

件时，可以通过查看实际的 API 响应来提高准确性。大多数 API 返回一个包含多个可能转录的 JSON 字符串，但若不强制要求给出完整响应时，recognition_google（）方法始终仅返回最可能的转录字符。

通过把 recognition_google（）中 True 参数改成 show_all 来给出完整响应。

```
>>> r.recognize_google(audio, show_all=True)

{'alternative': [

{'transcript':'the snail smell like old Beer Mongers'},

{'transcript':'the still smell of old beer vendors'},

{'transcript':'the snail smell like old beer vendors'},

{'transcript':'the stale smell of old beer vendors'},

{'transcript':'the snail smell like old beermongers'},

{'transcript':'destihl smell of old beer vendors'},

{'transcript':'the still smell like old beer vendors'},

{'transcript':'bastille smell of old beer vendors'},

{'transcript':'the still smell like old beermongers'},

{'transcript':'the still smell of old beer venders'},

{'transcript':'the still smelling old beer vendors'},

{'transcript':'musty smell of old beer vendors'},
```

{'transcript':'the still smell of old beer vendor'}

],'final':True}

可以看到，recognition_google（）返回了一个关键字为 'alternative' 的列表，指的是所有可能的响应列表。此响应列表结构会因 API 而异且主要用于对结果进行调试。

## ▌麦克风的使用

若要使用 SpeechRecognizer 访问麦克风则必须安装 PyAudio 软件包，请关闭当前的解释器窗口，进行以下操作：

**安装 PyAudio**

安装 PyAudio 的过程会因操作系统而异。

Debian Linux

如果使用的是基于 Debian 的 Linux（如 Ubuntu ），则可使用 apt 安装 PyAudio：

$ sudo apt-getinstall python-pyaudio python3-pyaudio

安装完成后可能仍需要启用 pip install pyaudio ，尤其是在虚拟情况下运行。

macOS

macOS 用户则首先需要使用 Homebrew 来安装 PortAudio，然后调用 pip 命令来安装 PyAudio。

$ brew install portaudio

$ pip install pyaudio

Windows

Windows 用户可直接调用 pip 来安装 PyAudio。

$ pip install pyaudio

安装测试

安装了 PyAudio 后可从控制台进行安装测试。

$ python –m speech_recognition

请确保默认麦克风打开并取消静音，若安装正常则应该看到如下所示的内容：

A moment of silence, please...

Setminimumenergy thresholdto600.4452854381937

Say something!

请对着麦克风讲话并观察 SpeechRecognition 如何转录你的讲话。

**Microphone 类**

请打开另一个解释器会话，并创建识一个别器类的例子。

>>> importspeech_recognitionassr

>>> r = sr.Recognizer()

此时将使用默认系统麦克风，而不是使用音频文件作为信号源。读者可通过创建一个 Microphone 类的实例来访问它。

```
>>> mic = sr.Microphone()
```

若系统没有默认麦克风（如在 RaspberryPi 上）或想要使用非默认麦克风，则需要通过提供设备索引来指定要使用的麦克风。读者可通过调用 Microphone 类的 list_microphone_names（）函数来获取麦克风名称列表。

```
>>> sr.Microphone.list_microphone_names()
['HDA Intel PCH: ALC272 Analog (hw:0,0)',
'HDA Intel PCH: HDMI 0 (hw:0,3)',
'sysdefault',
'front',
'surround40',
'surround51',
'surround71',
'hdmi',
'pulse',
'dmix',
'default']
```

注意：你的输出可能与上例不同。

list_microphone_names () 返回列表中麦克风设备名称的索引。在上面的输出中，如果要使用名为“front”的麦克风，该麦克风在列表中索引为 3，则可以创建如下所示的麦克风实例：

```
>>># This is just an example; do not run

>>> mic = sr.Microphone(device_index=3)
```

但大多数情况下需要使用系统默认麦克风。

**使用 listen（）获取麦克风输入数据**

准备好麦克风实例后，读者可以捕获一些输入。

就像 AudioFile 类一样，Microphone 是一个上下文管理器。可以使用 with 块中 Recognizer 类的 listen（）方法捕获麦克风的输入。该方法将音频源作为第一个参数，并自动记录来自源的输入，直到检测到静音时自动停止。

```
>>> withmicassource:

... audio = r.listen(source)

...
```

执行 with 块后请尝试在麦克风中说出“hello”。请等待解释器再次显示提示，一旦出现“>>>”提示返回就可以识别语音。

```
>>> r.recognize_google(audio)

'hello'
```

如果没有提示再次返回，可能是因为麦克风收到太多的环境噪音，请使用 Ctrl + C 中断这个过程，从而让解释器再次显示提示。

要处理环境噪声，可调用 Recognizer 类的 adjust_for_ambient_noise（）函数，其操作与处理噪音音频文件时一样。由于麦克风输入声音的可预测性不如音频文件，因此任何时间听麦克风输入时都可以使用此过程进行处理。

```
>>> withmicassource:

... r.adjust_for_ambient_noise(source)

... audio = r.listen(source)

...
```

运行上面的代码后稍等片刻，尝试在麦克风中说“hello”。同样，必须等待解释器提示返回后再尝试识别语音。

请记住，adjust_for_ambient_noise（）默认分析音频源中 1 秒钟长的音频。若读者认为此时间太长，可用 duration 参数来调整。

SpeechRecognition 资料建议 duration 参数不少于 0.5 秒。某些情况下，你可能会发现，持续时间超过默认的一秒会产生更好的结果。您所需要的最小值取决于麦克风所处的周围环境，不过，这些信息在开发过程中通常是未知的。根据我的经验，一秒钟的默认持续时间对于大多数应用程序已经足够。

**处理难以识别的语音**

尝试将前面的代码示例输入到解释器中，并在麦克风中输入一些无法理解的噪音。你应该得到这样的结果：

Traceback (most recentcalllast):

File"", line1,in

File"/home/david/real_python/speech_recognition_primer/venv/lib/python3.5/site-packages/speech_recognition/__init__.py", line858,inrecognize_google

ifnotisinstance(actual_result, dict)orlen(actual_result.get("alternative", [])) ==0:raiseUnknownValueError()

speech_recognition.UnknownValueError

无法被 API 匹配成文字的音频会引发 UnknownValueError 异常，因此要频繁使用 try 和 except 块来解决此类问题。API 会尽全力去把任何声音转成文字，如短咕噜声可能会被识别为“How”，咳嗽声、鼓掌声以及舌头咔哒声都可能会被转成文字从而引起异常。

**结语：**

本教程中，我们一直在识别英语语音，英语是 SpeechRecognition 软件包中每个 recognition _ *（）方法的默认语言。但是，识别其他语音也是绝对有可能且很容易完成的。要识别不同语言的语音，请将 recognition _ *（）方法的语言关键字参数设置为与所需语言对应的字符串。

作者：David Amos

原文链接：https://realpython.com/python-speech-recognition/