Contents

[第1章 语音识别模块 2](#_Toc492542415)

[1.1 语音识别技术 2](#_Toc492542416)

[1.1.1 语音基本概念 2](#_Toc492542417)

[1.1.2 语音识别技术基础 3](#_Toc492542418)

[1.1.3 语音识别系统模型 5](#_Toc492542419)

[1.2 Audioop 7](#_Toc492542420)

[1.3 Pyaudio 7](#_Toc492542421)

[1.3.1 具体功能 8](#_Toc492542422)

[1.3.2 关键接口 9](#_Toc492542423)

[1.4 Pocketsphinx 10](#_Toc492542424)

[1.4.1 CMUSphinx编译与安装 10](#_Toc492542425)

[1.4.2 CMUSphinx中文语言模型和声学模型的使用 11](#_Toc492542426)

[1.4.3 CMUSphinx Python接口 12](#_Toc492542427)

[第2章 图像识别 13](#_Toc492542428)

[2.1 安装Tensorflow 13](#_Toc492542429)

[2.2 安装Tensorflow模型 14](#_Toc492542430)

[2.3 Tensorflow识别图像 14](#_Toc492542431)

[第3章 其它模块 14](#_Toc492542432)

[3.1 Apscheduler 14](#_Toc492542433)

[3.2 atexit 15](#_Toc492542434)

[3.3 Ctypes 15](#_Toc492542435)

[3.4 tempfile 15](#_Toc492542436)

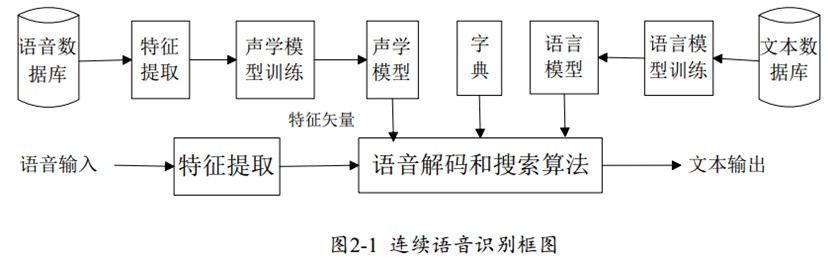
[3.5 Pyinotify 15](#_Toc492542437)

[3.6 pyyaml 16](#_Toc492542438)

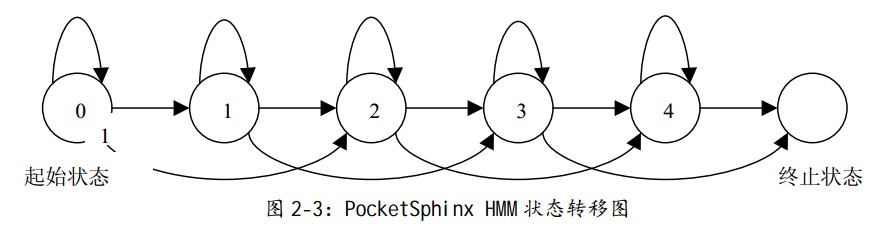
1. 语音识别模块
   1. 语音识别技术
      1. 语音基本概念

* ****音素：****语音的最小单位
  + 从声学性质来看：音素是从音质角度划分出来的最小语音单位。
  + 从生理性质来看：一个发音动作形成一个音素。如“ma”包含“m”“a”两个发音动作，是两个音素。相同发音动作发出的音就是同一音素，不同发音动作发出的音就是不同音素。如“ma-mi”中，两个“m”发音动作相同，是相同音素，“a”“i”发音动作不同，是不同音素。
  + **汉语音素：**汉语包括10个元音，22个辅音，总共有32个。
  + **英语音素：**英语国际音标共有48个音素，其中元音音素20个，辅音音素28个。
* ****音节：****听觉能感受到的最自然的语音单位由一个或几个音素按一定规律组合而成
  + **汉语音节:**
* 汉语中一个汉字就是一个音节，每个音节由声母、韵母和声调三个部分组成。拼音就是把声母、韵母、声调急速连续拼合并加上声调而成为一个音节。如：q-i-áng→qiáng（强）。
* 汉语普通话中的无调音节（不做音调区分）共有400个音节。一个音节至少有一个音素，至多有四个音素。
  + **英语音节：**
* 英语音节是读音的基本单位，任何单词的读音，都是分解为一个个音节朗读。
* 英语中一个元音音素可构成一个音节，一个元音音素和一个或几个辅音音素结合也可以构成一个音节。英语的词有一个音节，两个音节或多个音节的。一个音节叫单音节词，两个音节叫双音节词，三个音节以上叫多音节。如：take拿，ta'ble ，po‘ta'to， con’gra‘tu’la'tion
* 元音音素是构成音节的主体，辅音是音节的分界线。每个元音音素都可以构成一个音节。
* 语音：一个或多个音素构成音节，一个或多个音节构成一个单词，单词构成语音
  + 语音的特点：
* 语音是一个连续的音频流，它是由大部分的稳定态和部分动态改变的状态混合构成。
* 一个单词实际发声的波形取决于很多因素，如音素、音素上下文、说话者、语音风格等。
* 协同发音：指的是一个音受前后相邻音的影响而发生变化。从发声机理上看就是人的发声器官在一个音转向另一个音时其特性只能渐变，从而使得后一个音的频谱与其他条件下的频谱产生差异。使得音素的感知与标准不一样，需要根据上下文来辨别音素。
* 从某种程度上面讲语音的描述都是基于概率的（频谱），意味着在语音单元或者单词之间并没有确定的边界，语音识别技术没办法到达100%的准确率。
  + 语音上下文依赖和左右上下文依赖：
* 左右上下文依赖：由于协同发音的存在需要将一个音素划分为几个亚音素单元。如：数字“three”，音素的第一部分与在它之前的音素存在关联，中间部分是稳定的部分，而最后一部分则与下一个音素存在关联，这就是为什么在用HMM模型做语音识别时，选择音素的三状态HMM模型。
* 上下文依赖：音素会被放在上下文中考虑，这样就形成了三元音素或者多元音素。但它与亚音素不同，他们在波形中匹配时长度还是和单一音素一样，它比亚音素要复杂得多，它是一个可以被决策树或者其他方式来定义的复杂函数。
* ****语料库：****通常指为语言研究收集的、用电子形式保存的语言材料
  + 在统计[自然语言](http://lib.csdn.net/base/nlp" \t "_blank" \o "自然语言理解和处理知识库)处理中实际上不可能观测到大规模的语言实例。所以简单的用文本作为替代，并把文本中的上下文关系作为现实世界中语言的上下文关系的替代品。
  + 把一个文本集合称为语料库，当有几个这样的文本集合的时候，称之为语料库集合。由自然出现的书面语或口语的样本汇集而成，用来代表特定的语言或语言变体。如：说话的时候的句子、一些文学作品的语句段落、报刊杂志上出现过的语句段落等等在现实生活中真实出现过的语言材料等。
    1. 语音识别技术基础
* 语音识别技术：
  + 定义：让机器通过识别和理解过程把语音信号转变为相应的文本或命令的技术
* 其是一种模式识别过程：未知语音的模式与已知语音的参考模式逐一进行比较，最佳匹配的参考模式被作为识别结果。目的是让机器能听懂人说话并作出相应的动作。
* 目前大多数语音识别技术是基于统计模式的，从语音产生机理来看，语音识别可以分为语音层和语言层两部分。
  + 主流[算法](http://lib.csdn.net/base/datastructure" \t "_blank" \o "算法与数据结构知识库)：主要有基于动态时间规整(DTW)算法、基于非参数模型的矢量量化(VQ)方法、基于参数模型的隐马尔可夫模型(HMM)的方法、基于人工神经网络(ANN)和支持向量机等。
  + 语音识别技术分类：
* 根据对说话人的依赖程度分为：
  + - * 特定人语音识别（SD）：只能辨认特定使用者的语音，在训练后才能使用。
      * 非特定人语音识别（SI）：可辨认任何人的语音，无须训练。
* 根据对说话方式的要求分为：
  + - * 孤立词识别：每次只能识别单个词汇。
      * 连续语音识别：用者以正常语速说话，即可识别其中的语句。
  + 语音识别产品分类：语音识别芯片、语音识别软件。
* 语音识别一般过程：
  + 录制语音波形，再把波形通过静音分割为多个表达部分。
  + 然后去识别每个表达部分所表达的意思。为了达到这个目的，需要用单词的所有可能组合去匹配这段音频，然后选择匹配度最高的组合。
* 语音识别基础概念：
  + 语音特征：描述一个语音需要的参数个数非常多，需要进行降维优化。可以用帧frames去分割语音波形，每帧大概10ms，然后每帧提取可以代表该帧语音的39个数字，这39个数字也就是该帧语音的特征，用特征向量来表示。提取方法都是由频谱衍生出来的。
  + 语音模型：用来描述一些口语的共同属性，在实际应用中音频模型就是三态高斯混合模型，即它就是一个最有可能的特征向量
* 声学模型：一个声学模型包含每个音素的声学属性，其包括不依赖于上下文的属性和依赖于上下文的属性。
* **语音学字典：**包含了从单词到音素之间的映射。字典并不是描述单词到音素之间的映射的唯一方法，也可以通过运用[机器学习](http://lib.csdn.net/base/machinelearning)算法去学习得到一些复杂的函数去完成映射功能。
* 语言模型：语言模型是用来约束单词搜索的，它定义了哪些词能跟在上一个已经识别的词的后面，这样就可以为匹配过程排除一些不可能的单词。为了达到比较好的识别准确率，语言模型必须能够很好的约束空间搜索，也就是说可以更好的预测下一个词。语言模型可以包含更小的块，例如亚单词，甚至音素。但是这种情况，识别准确率将会低于基于单词的语言模型。
  + **匹配**[算法](http://lib.csdn.net/base/datastructure)**：**语音识别需要对所有的特征向量和所有的模型做比较匹配，这是一个非常耗时的工作。
* 特征、模型和搜索算法三部分构成了一个语音识别系统。如果需要识别不同的语言，那么就需要修改这三个部分。很多语言，都已经存在声学模型，字典，甚至大词汇量语言模型可供下载了。
  + 1. 语音识别系统模型

通常由声学模型和语言模型两部分组成，分别对应于语音到音节概率的计算和音节到字概率的计算。



* ****预处理模块：****
  + 对输入的原始语音信号进行处理，滤除掉其中的不重要的信息以及背景噪声。
  + 语音信号的端点检测，找出语音信号的始末。
  + 语音分帧，近似认为在10-30ms内是语音信号是短时平稳的，将语音信号分割为一段一段进行分析以及预加重（提升高频部分）等处理。
* ****特征提取：****
  + 去除语音信号中对于语音识别无用的冗余信息，保留反映语音本质特征的信息。也就是提取出反映语音信号特征的关键特征参数形成特征矢量序列，以便用于后续处理。
  + 提取特征的方法：一般都是由频谱衍生出来
* Mel频率倒谱系数（MFCC）参数：MFCC的计算首先用FFT将时域信号转化成频域，之后对其对数能量谱用依照Mel刻度分布的三角滤波器组进行卷积，最后对各个滤波器的输出构成的向量进行离散余弦变换DCT，取前N个系数。
* Sphinx采用MFCC特征提取，首先用帧frames去分割语音波形，每帧大概10ms，然后每帧提取可以代表该帧语音的39个数字，这39个数字也就是该帧语音的MFCC特征，用特征向量来表示。
* ****声学模型训练：****根据训练语音库的特征参数训练出声学模型参数。在识别时可以将待识别的语音的特征参数同声学模型进行匹配，得到识别结果
  + 声学模型的建模单元：可以是音素，音节，词等各个层次。对于小词汇量的语音识别系统，可以直接采用音节进行建模。而对于词汇量偏大的识别系统，一般选取音素，即声母，韵母进行建模。识别规模越大，识别单元选取的越小。
  + 隐马尔可夫模型HMM声学模型建模：
* 人的言语过程实际上就是一个双重随机过程，语音信号本身是一个可观测的时变序列，是由大脑根据语法知识和言语需要（不可观测的状态）发出的音素的参数流（发出的声音）。
* HMM合理地模仿了双重随机过程：用HMM刻画语音信号需作出两个假设，一是内部状态的转移只与上一状态有关，另一是输出值只与当前状态（或当前的状态转移）有关，这两个假设大大降低了模型的复杂度。对语音信号的时间序列结构建立统计模型，将其看作一个数学上的双重随机过程:
  + - * 一个是用具有有限状态数的Markov链来模拟语音信号统计特性变化的隐含的随机过程。
      * 另一个是与Markov链的每一个状态相关联的外界可见的观测序列（通常就是从各个帧计算而得的声学特征）的随机过程。
* HMM通常是用从左向右单向、带自环、带跨越的拓扑结构来对识别基元建模，一个音素就是一个三至五状态的HMM，一个词就是构成词的多个音素的HMM串行起来构成的HMM，而连续语音识别的整个模型就是词和静音组合起来的HMM。

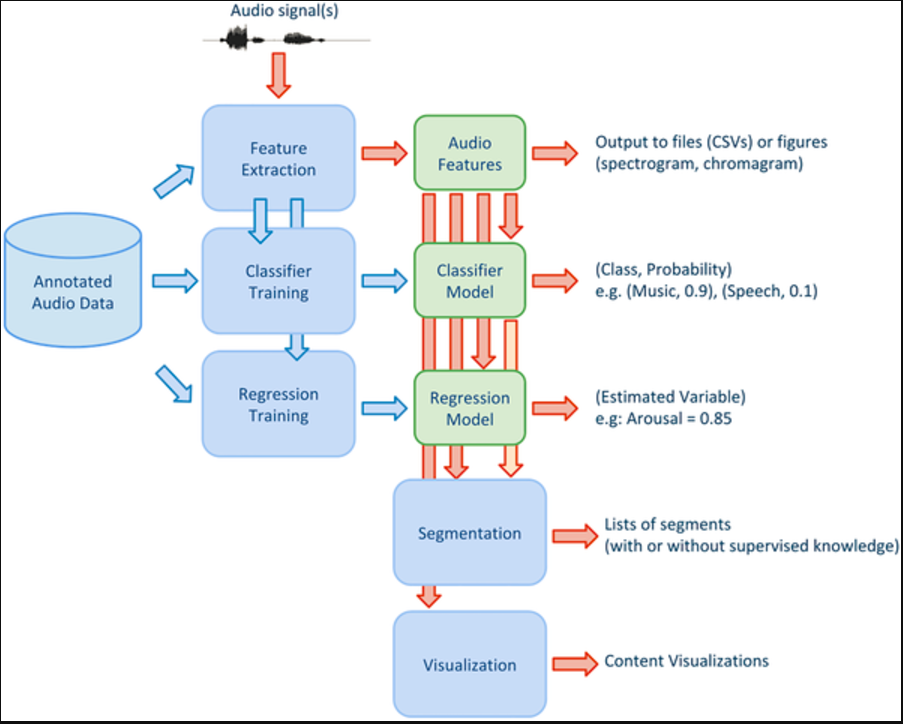


* ****语言模型训练：****用来计算一个句子出现概率的概率模型
  + 它主要用于决定哪个词序列的可能性更大，或在出现了几个词的情况下预测下一个即将出现的词语的内容。语言模型是用来约束单词搜索的。它定义了哪些词能跟在上一个已经识别的词的后面，这样就可以为匹配过程排除一些不可能的单词。
  + 语言建模能够有效的结合汉语语法和语义的知识，描述词之间的内在关系，从而提高识别率，减少搜索范围。语言模型分为三个层次：字典知识，语法知识，句法知识。
  + 对训练文本[数据库](http://lib.csdn.net/base/mysql)进行语法、语义分析，经过基于统计模型训练得到语言模型。语言建模方法主要有基于规则模型和基于统计模型两种方法。
* N-Gram模型基于概率统计，它包含了单词序列的统计，基于这样一种假设，第n个词的出现只与前面N-1个词相关，而与其它任何词都不相关，整句的概率就是各个词出现概率的乘积。这些概率可以通过直接从语料中统计N个词同时出现的次数得到。常用的是二元的Bi-Gram和三元的Tri-Gram。
* Sphinx中是采用二元语法和三元语法的统计语言概率模型，也就是通过前一个或两个单词来判定当前单词出现的概率 P(w2| w1)，P(w3| w2, w1)。
* ****语音解码和搜索算法：****
  + **解码器：**指语音技术中的识别过程
* 针对输入的语音信号，根据己经训练好的HMM声学模型、语言模型及字典建立一个识别网络，根据搜索算法在该网络中寻找最佳的一条路径，这个路径就是能够以最大概率输出该语音信号的词串，这样就确定这个语音样本所包含的文字了。
* 解码操作即指搜索算法，指在解码端通过搜索技术寻找最优词串的方法。
  + 连续语音识别搜索：寻找一个词模型序列以描述输入语音信号，从而得到词解码序列。搜索所依据的是对公式中的声学模型打分和语言模型打分。
* 在实际使用中，往往要依据经验给语言模型加上一个高权重，并设置一个长词惩罚分数。
* 连续语音识别系统的识别结果是一个词序列。解码实际上是对词表的所有词反复搜索。词表中词的排列方式会影响搜索的速度，而词的排列方式就是字典的表示形式。Sphinx系统中采用音素作为声学训练单元，通常字典就用来记录每个单词由哪些个音素组成，也可以理解为对每个词的发音进行标注。
  + **搜索算法：**
* Viterbi搜索算法：基于动态规划的Viterbi算法在每个时间点上的各个状态，计算解码状态序列对观察序列的后验概率，保留概率最大的路径，并在每个节点记录下相应的状态信息以便最后反向获取词解码序列。Viterbi本质上是一种动态规划算法，该算法遍历HMM状态网络并保留每一帧语音在某个状态的最优路径得分。
* **N-best搜索和多遍搜索：**为在搜索中利用各种知识源，通常要进行多遍搜索，第一遍使用代价低的知识源（如声学模型、语言模型和音标词典），产生一个候选列表或词候选网格，在此基础上进行使用代价高的知识源（如4阶或5阶的N-Gram、4阶或更高的上下文相关模型）的第二遍搜索得到最佳路径。
  1. Audioop

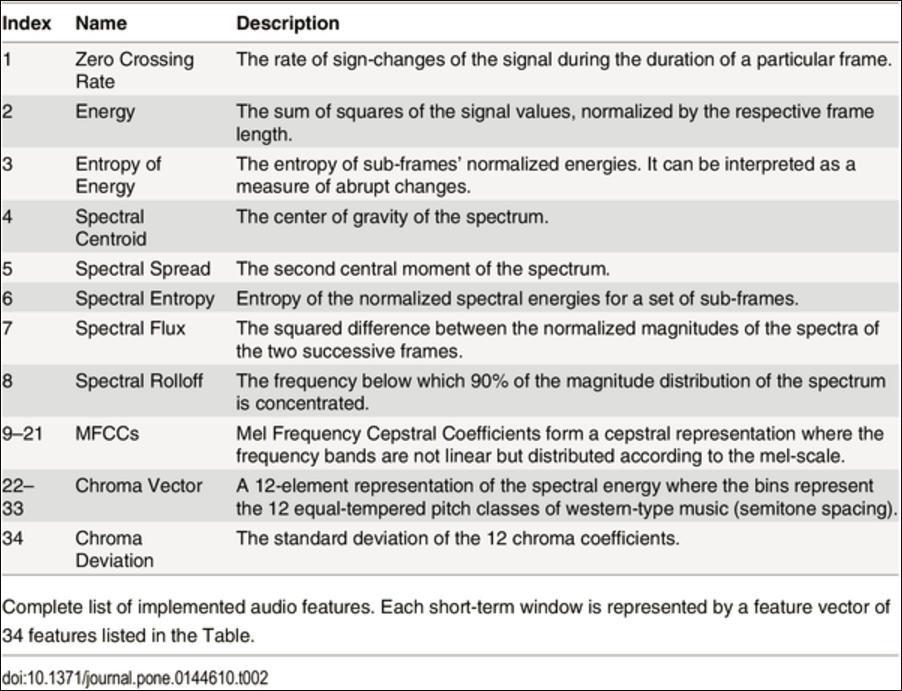
The module contains some useful operations on sound fragments. It operates on sound fragments consisting of signed integer samples 8, 16 or 32 bits wide, stored in Python strings. This is the same format as used by the [al](https://docs.python.org/2/library/al.html#module-al) and [sunaudiodev](https://docs.python.org/2/library/sunaudio.html" \l "module-sunaudiodev" \o "sunaudiodev: Access to Sun audio hardware. (deprecated) (SunOS)) modules. All scalar items are integers, unless specified otherwise.nnn

* audioop.rms(fragment, width)
  + Return the root-mean-square of the fragment, i.e. sqrt(sum(S\_i^2)/n).
  + This is a measure of the power in an audio signal.
  1. Pyaudio

PyAudio为PortAudio提供了Pythonbindings，即跨平台音频I / O库。Pyaudio功能强大的处理音频库，功能包括：提取特征、训练并且使用分类器、语音分割功能、内容关系可视化。



* alsa 音频库的移植：主要是alsa-lib编译出来的库libasound
  + 1. 具体功能
* 特征提取(feature extraction)：
  + 短时间特征：时域特征（特征1-3）直接提取、频域特征（特征4-34，MFCC除外）由傅里叶转换而来。
  + 中等长度特征和特别长的特征：中等长度的调用短时间的特征，并进行统计运算1-10分钟左右长度，特别长的特征调用中等长度特征然后做统计运算。
  + 速度有关的特征。



* 语音分割：
  + 有监督的语音分割：需要有提供一些材料，例如，已经训练好的分类器。为此，此库提供了两种[算法](http://lib.csdn.net/base/datastructure" \t "_blank" \o "算法与数据结构知识库)：分类的方法和隐马尔科夫的算法。
* 固定长度的分割算法：audioSegmentation.py的函数mtFileClassfication()是主要的
  + - * 切分一个语音段为一个连续的中等长度的段并且提取min-term features，使用mtFeatureExtraction()。
      * 使用已经训练好的模型来进行分类。
      * 只要两个段相连并且都是同一个类别，就将两个段进行合并。
      * 将数据进行可视化。
* 注意事项，一个.segment的文件必须要有plotSegmentationResults()用来画图可视化数据
* 隐马尔科夫模型分割算法（HMM-based segmentation）
  + - * 用户需要提供已经标注过的数据，包括：段开始点，段结束点，段类别。
      * 一个语音文件配一个标注的配置文件，两者构成训练集。
      * TrainHMM\_fromFile()和TrainHMM\_fromDir()是两个训练的函数。
      * hmmSegmentation()函数用来使用HMM算法。
      * plotSegmentationResult()可视化并且计算正确率。
  + 无监督的语音分割：无监督的或者半监督的，不需要提供知识准备，主要的例子是静音检测，语音聚类，语音缩略图。
* 静音检测
  + - * 短时间特征提取。
      * 训练SVM，在高能帧和低能帧中，特别的，10%的高能帧和10%的低能帧用在训练SVM模型。
      * 将SVM分类器用在全部的语音中，得到一串概率值，对应此帧是个时间帧（audio event）的概率。
      * 一个动态的阈值被用来探测段是不是活跃的。
* 语音聚类（speaker diarization）
* 语音缩略图(audio thumbnailing)
* 回归(regression)：将语音信号映射到一个回归值。
* 可视化：给定语音，将内容可视化
  + 1. 关键接口

1. p=pyaudio.PyAudio()
2. stream=p.open(format=pyaudio.paInt16,#格式
3. channels=2,#通道
4. rate=44100,#采样率
5. input=True,
6. frames\_per\_buffer=1024)
7. print "recording"
8. frames=[]
9. for i in range(0,int(44100/1024\*10)):#录音秒数
10. data=stream.read(1024)
11. frames.append(data)
12. print "recording ok"
13. stream.stop\_stream()
14. stream.close()
15. p.terminate()
    1. Pocketsphinx

Sphinx是由美国卡内基梅隆大学开发的大词汇量、非特定人、连续英语语音识别系统。Sphinx版本：

1. Sphinx-2采用半连续隐含马尔可夫模型建模，识别精度要低于其它的译码器。
2. PocketSphinx计算量和体积都很小的[嵌入式](http://lib.csdn.net/base/embeddeddevelopment" \t "_blank" \o "嵌入式开发知识库)语音识别引擎。在Sphinx-2的基础上针对嵌入式系统的需求修改、优化而来，识别精度和Sphinx-2差不多。
3. Sphinx-3是CMU高水平的大词汇量语音识别系统，采用连续隐含马尔可夫模型建模。支持高精度模式、快速搜索模式。
4. Sphinx-4是由[Java](http://lib.csdn.net/base/java" \t "_blank" \o "Java 知识库)语言编写的，采用连续的隐含马尔可夫模型建模，它在模块化、灵活性和[算法](http://lib.csdn.net/base/datastructure)方面做了改进，采用新的搜索策略，支持各种不同的语法和语言模型、听觉模型和特征流，创新的算法允许多种信息源合并成一种更符合实际语义的优雅的知识规则。具有高度的可移植性，允许多线程技术和高度灵活的多线程接口。
   * 1. ****CMUSphinx编译与安装****

以Ubuntu操作系统为例

* 依赖包安装：

rocky@intel-desktop:~$ sudo apt-get install -y perl python emacs git-core python-dev python-pip bison libasound2-dev libportaudio13-dev python-pyaudio libatlas-base-dev python-pymad subversion autoconf libtool automake gfortran g++

* ****CMUSphinx安装包下载安装：****<https://sourceforge.net/projects/cmusphinx/files/>
  + **安装Sphinxbase：**主要完成的是语音信号的特征提取

rocky@intel-desktop:~$ wget http://downloads.sourceforge.net/project/cmusphinx/sphinxbase/0.8/sphinxbase-0.8.tar.gz

rocky@intel-desktop:~$ tar -xzf sphinxbase-0.8.tar.gz

rocky@intel-desktop:~$ cd sphinxbase-0.8

rocky@intel-desktop:~$ ./autogen.sh

rocky@intel-desktop:~$./configure --enable-fixed

rocky@intel-desktop:~$ make && make install

rocky@intel-desktop:~$ ldconfig

* ldconfig将 /etc/ld.so.conf列出的路径下的库文件缓存到/etc/ld.so.cache以供使用，系统是通过读取这个库列表文件来加快搜索链接路径的速度的。
  + 安装Pocketsphinx：主要是进行语音识别

rocky@intel-desktop:~$ wget http://downloads.sourceforge.net/project/cmusphinx/pocketsphinx/0.8/pocketsphinx-0.8.tar.gz

rocky@intel-desktop:~$ tar -zxvf pocketsphinx-0.8.tar.gz

rocky@intel-desktop:~$ cd pocketsphinx-0.8/

rocky@intel-desktop:~$./configure && make && make install

* + 安装CMUclmtk：语言模型训练工具，若采用现成模型，那就不需要安装这个工具

rocky@intel-desktop:~$ pip install cmuclmtk

# or

rocky@intel-desktop:~$ svn co https://svn.code.sf.net/p/cmusphinx/code/trunk/cmuclmtk/  
rocky@intel-desktop:~$ cd cmuclmtk/  
rocky@intel-desktop:~$./autogen.sh && make && make install

* + 安装Sphinxtrain：声学模型训练工具，若采用现成模型，那就不需要安装这个工具

rocky@intel-desktop:~$ wget https://jaist.dl.sourceforge.net/project/cmusphinx/sphinxtrain/1.0.8/sphinxtrain-1.0.8.tar.gz

rocky@intel-desktop:~$ tar -xzf sphinxtrain-1.0.8.tar.gz

rocky@intel-desktop:~$ cd sphinxtrain-1.0.8

rocky@intel-desktop:~$./configure && make

* 没有make install的安装过程，因为sphinxtrain只是包含了用来训练声学模型的Perl和[Python](http://lib.csdn.net/base/python" \t "_blank" \o "Python知识库)脚本
  + 1. ****CMUSphinx中文语言模型和声学模型的使用****
* 获得语言模型和声学模型：
  + 源代码自带：声学模型（pocketsphinx-0.7/model/hmm/zh/tdt\_sc\_8k），语言模型（pocketsphinx-0.7/model/lm/zh\_CN ），hmm表示隐马尔可夫声学模型，lm表示language model语言模型。在安装时被拷贝到/usr/local/share/pocketsphinx/model目录。
  + **网上下载：** [sphinx官网](%09http:/sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/)提供了各种语言的声学模型和语言模型的下载：
* 声学模型：zh\_broadcastnews\_16k\_ptm256\_8000.tar.bz2
* 语言模型：zh\_broadcastnews\_64000\_utf8.DMP
* 字典文件：zh\_broadcastnews\_utf8.dic
  + 行训练得到：因为每个人的声音不一样，另外应用也不一样（所需词汇类别和不同），所以别人的模型可能在自己的语音下识别准确率不高，这样就需要训练自己的声学模型和语言模型（CMU也提供一个改进现有声学模型的方法）。
* pkg-config --variable=modeldir pocketsphinx这个命令来获得模型的安装目录。也就是/usr/local/share/pocketsphinx/model
* ****使用语言模型和声学模型：（示例为网上下载的方式）****
  + 下载资源：
* 声学模型：zh\_broadcastnews\_16k\_ptm256\_8000.tar.bz2
* 语言模型：zh\_broadcastnews\_64000\_utf8.DMP
* 字典文件：zh\_broadcastnews\_utf8.dic
  + 先把以上三个文件放在同一个目录下，再把声学模型解压：

rocky@intel-desktop:~$ tar -xjf zh\_broadcastnews\_16k\_ptm256\_8000.tar.bz2

解压后会在当前文件夹生成一个包含以下八个文件的文件夹：

1. zh\_broadcastnews\_ptm256\_8000/
2. ├── feat.params   //HMM模型的特征参数
3. ├── mdef   //模型定义文件（为每个即将进行训练的HMM的每一状态定义一个独特的数字标识）
4. ├── means  //混合高斯模型的均值
5. ├── mixture\_weights   //混合权重
6. ├── noisedict    //噪声也就是非语音字典
7. ├── sendump  //用来从声学模型中获取mixture\_weights文件的？？
8. ├── transition\_matrices  //HMM模型的状态转移矩阵
9. └── variances  //混合高斯模型的方差
   * 然后执行：

rocky@intel-desktop:~$ pocketsphinx\_continuous -hmm zh\_broadcastnews\_ptm256\_8000/ -lm zh\_broadcastnews\_64000\_utf8.DMP -dict zh\_broadcastnews\_utf8.dic

* -hmm选项指定声学模型文件夹 –lm选项指定语言模型 –dict选项指定字典文件；
  + pocketsphinx会分析传入的文件取得模型和识别的参数，然后显示出来，等到显示：

rocky@intel-desktop:~$ Warning: Could not find Mic element

READY....

到这里你就可以喊话让它识别了。对着麦克风讲话结束后在屏幕上以文字形式显示识别结果。在麦克风输入时，两者的最长语音时间为5秒，前者是通过说话人的正常语句停顿作为断句的分割点，并将停顿之前的语音信号作为一个处理段。

* + 1. ****CMUSphinx Python接口****

/home/rocky/Downloads/pocketsphinx-0.8/python/pocketsphinx.pyx

1. cdef class Decoder:
2. """PocketSphinx speech decoder.To initialize the PocketSphinx decoder, pass a list of keyword arguments to the constructor::
3. d = pocketsphinx.Decoder(hmm='/path/to/acoustic/model',

lm='/path/to/language/model',

dict='/path/to/dictionary',

beam='1e-80')

1. If no arguments are passed, the default acoustic and language models will be loaded, which may be acceptable for general English speech. Any arguments supported by the PocketSphinx decoder are allowed here. Only the most frequent ones are described below.
2. @param boxed: Boxed pointer from GStreamer containing a decoder;@type boxed: PyGBoxed
3. @param hmm: Path to acoustic model directory;@type hmm: str
4. @param dict: Path to dictionary file;@type dict: str
5. @param lm: Path to language model file;@type lm: str
6. @param jsgf: Path to JSGF grammar file;@type jsgf str

* 初始化解码器：

1. Decoder = pocketsphinx.Decoder()

* 解码语音：

1. data = fp.read()
2. “””Prepare the decoder to recognize an utterance: @param uttid: Identifier to give to this utterance. @type uttid: str”””
3. self.\_decoder.start\_utt(uttid=None))
4. """ Process (decode) some audio data: @param data: Audio data to process. This is packed binary data, which consists of single-channel, 16-bit PCM audio, at the sample rate specified when the decoder was initialized; @type data: str; @param no\_search: Buffer the data without actually processing it (default is to process the data as it is received); @type no\_search: bool; @param full\_utt: This block of data is an entire utterance. Processing an entire utterance at once may improve recognition, particularly for the first utterance passed to the decoder; @type full\_utt: bool"""
5. self.\_decoder.process\_raw(data, no\_search=False, full\_utt=True))
6. #Finish processing an utterance.
7. self.\_decoder.end\_utt()
8. """Get a hypothesis string. This function returns the text which has been recognized so far, or, if C{end\_utt()} has been called, the final recognition result: @return: Hypothesis string, utterance ID, recognition score; @rtype: (str, str, int) """
9. result =self.\_decoder.get\_hyp()
10. words = [result[0]]
11. 图像识别
    1. 安装Tensorflow

使用github上已经编译好的Tensorflow [tensorflow-on-raspberry-pi](https://github.com/samjabrahams/tensorflow-on-raspberry-pi)

# For Python 2.7

rocky@intel-desktop:~$ wget https://github.com/samjabrahams/tensorflow-on-raspberry-pi/releases/download/v1.1.0/tensorflow-1.1.0-cp27-none-linux\_armv7l.whl

rocky@intel-desktop:~$ sudo pip install tensorflow-1.1.0-cp27-none-linux\_armv7l.whl

#reinstall the mock library to keep it from throwing an error when we import TensorFlow

rocky@intel-desktop:~$ sudo pip uninstall mock

rocky@intel-desktop:~$ sudo pip install mock

* 1. 安装Tensorflow模型

参照github：[pi\_examples](https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/pi_examples)使用[inception](https://github.com/tensorflow/models/tree/master/inception)的模型（基于ImageNet数据集）。它可以完成物体识别，直接使用预训练好的模型。训练模型可是个费时费力的工作。ImageNet数据集包含约120万张训练图像、5万张验证图像和10万张测试图像，分为1000个不同的类别，用于机器学习中训练图像识别系统

# Install libjpeg, so we can load image files

rocky@intel-desktop:~$ sudo apt-get install -y libjpeg-dev

# To download the example model you'll need, run these commands

rocky@intel-desktop:~$ curl https://storage.googleapis.com/download.tensorflow.org/models/inception\_dec\_2015\_stripped.zip

-o /tmp/inception\_dec\_2015\_stripped.zip

rocky@intel-desktop:~$ mkdir -p /home/rocky/tensorflow/model

rocky@intel-desktop:~$ unzip /tmp/inception\_dec\_2015\_stripped.zip -d /home/robot/tensorflow/model

# git clone install models code

rocky@intel-desktop:~$ cd /usr/local/lib/python2.7/dist-packages/tensorflow/

rocky@intel-desktop:~$ git clone https://github.com/tensorflow/models.git

* 1. Tensorflow识别图像

cd /usr/local/lib/python2.7/dist-packages/tensorflow/models/tutorials/image/imagenet

#--model\_dir 指定模型数据存放的目录

python classify\_image.py --model\_dir /home/robot/tensorflow/model --image\_file /home/robot /tensorflow/test-images/mobike.jpg

1. 其它模块
   1. Apscheduler

APScheduler是基于Quartz的一个Python定时任务框架，实现了Quartz的所有功能，使用起来十分方便。提供了基于日期、固定时间间隔以及crontab类型的任务，并且可以持久化任务。基于这些功能，可以很方便的实现一个python定时任务系统。

1. from apscheduler.schedulers.background import BackgroundScheduler
2. def tick():
3. print('Tick! The time is: %s' % datetime.now())
4. scheduler = BackgroundScheduler()
5. #非阻塞调度，间隔一段时间执行一次
6. scheduler.add\_job(tick, 'interval', seconds=3)　　#间隔3秒钟执行一次
7. scheduler.start() #这里的调度任务是独立的一个线程
8. #非阻塞调度，在指定的时间执行一次
9. scheduler.add\_job(tick, 'date', run\_date='2016-02-14 15:01:05')　　#在指定的时间，只执行一次
10. scheduler.start() #这里的调度任务是独立的一个线程
11. #非阻塞的方式，采用cron的方式执行
12. scheduler.add\_job(tick, 'cron', day\_of\_week='6', second='\*/5')
13. scheduler.start() #这里的调度任务是独立的一个线程
    1. atexit

atexit模块主要的作用就是在程序即将结束之前执行的代码，atexit模块使用register函数用于注册程序退出时的回调函数，然后在回调函数中做一些资源清理的操作。

* 1. Ctypes

是Python的一个外部库，提供和C语言兼容的数据类型，可以很方便地调用C DLL中的函数Objdll = ctypes.cdll.LoadLibrary("dllpath")

* 1. tempfile

tempfile 临时文件(夹)操作。tempfile.NamedTemporaryFile函数的行为与tempfile.TemporaryFile类似，只不过它多了一个delete参数，用于指定类文件对象close或者被del之后，是否也一同删除磁盘上的临时文件（当delete = True的时候，行为与TemporaryFile一样）。

* 1. Pyinotify

Pyinotify是一个Python模块，用来监测文件系统的变化。，其中他们可以监控文件夹内的文件的创建，修改，读取，删除等系列操作

1. import os
2. from pyinotify import WatchManager, Notifier, \
3. ProcessEvent,IN\_DELETE, IN\_CREATE,IN\_MODIFY
5. class EventHandler(ProcessEvent):
6. """事件处理"""
7. def process\_IN\_CREATE(self, event):
8. print  "Create file: %s " %  os.path.join(event.path,event.name)
10. def process\_IN\_DELETE(self, event):
11. print  "Delete file: %s " %  os.path.join(event.path,event.name)
13. def process\_IN\_MODIFY(self, event):
14. print  "Modify file: %s " %  os.path.join(event.path,event.name)
16. def FSMonitor(path='.'):
17. wm = WatchManager()
18. mask = IN\_DELETE | IN\_CREATE |IN\_MODIFY
19. notifier = Notifier(wm, EventHandler())
20. wm.add\_watch(path, mask,auto\_add=True,rec=True)
21. print 'now starting monitor %s'%(path)
22. while True:
23. try:
24. notifier.process\_events()
25. if notifier.check\_events():
26. notifier.read\_events()
27. except KeyboardInterrupt:
28. notifier.stop()
29. break
31. if \_\_name\_\_ == "\_\_main\_\_":
32. FSMonitor('/home/firefoxbug')
    1. pyyaml

pyyaml:YAML是一个堪比XML，JSON数据格式的更加方便，简洁的，易于人眼阅读的序列化数据格式