Chapter 20
Automatic Speech Recognition

Abstract There are basically two application modes for automatic speech recognition (ASR): using speech as spoken input or as knowledge source. Spoken input addresses applications like dictation systems and navigation (transactional) systems. Using speech as a knowledge source has applications like multimedia indexing systems. The chapter presents the stages of speech recognition process, resources of ASR, role and functions of speech engine—like, Julius speech recognition engine, voice-over web resources, ASR algorithms, language model and acoustic models—like HMM (hidden Markov models). Many open-source tools like—Kaldi speech recognition toolkit, CMU-Sphinx, HTK, and Deep speech tools' introduction, and guidelines for their usages are presented. These tools have interfaces with high-level languages like C/C++ and Python. The is followed with chapter summary and set of exercises.

Keywords Automatic speech recognition · ASR · Multimedia indexing · ASR resources · Language model · Acoustic model · Julius · Kaldi · CMU-Sphinx · HTK · Deep tools

20.1 Introduction

Speech has long been viewed as the future of computer interfaces, promising significant improvements in ease of use over the traditional keyboard and mouse. There are basically two application modes that exist for speech recognition: 1. Speech as *spoken input* to computers, and 2. The speech is used as *data* or *knowledge* source. The first application mode comprises the potential applications as, dictation systems, navigation, and transactional systems.

In the application of *dictation*, a system transcribes the spoken words into written text, and for dictating letters, reports, business correspondence, or e-mail messages, to the computer/machine.

The speech can be used in the form of commands to *navigate* around the applications, for example, selection of main application, then its one of the sub-applications, and sub-sub-application, till you reach to the command to execute the final application.

© Springer Nature India Private Limited 2020

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_20

The speech can be used for *transactional* applications, i.e., to use speech in the form of command or command sequence to cause a transaction to be performed. For example, the speech-based transactions can be purchase of stock, book a flight ticket, reserve an itinerary for tour, or doing the fund transfer.

The second mode of speech application, i.e., speech as a knowledge source has applications like meeting capture, and knowledge management. These applications are basically multimedia indexing systems that use speech recognition to transcribe words verbatim from an audio file into text. Subsequently, the IR (Information Retrieval) techniques are applied to the transcript to create an index with time offsets into the audio. Users can access the index using text keywords to search a collection of audio or video documents.

We understand both written and spoken language, and also know that skill of reading is learned much later. We now focus on spoken language. The problem of understanding the spoken language can be divided into major parts discussed below [6].

Phonological

The phonological processing step relates sounds to the words we recognize. Phone is smallest unit of sound, and the phones are aggregated into word sounds.

Morphological

This is lexical knowledge, which relates to word construction from basic units called morphemes. A morpheme is the smallest unit of meaning, for example, the construction of *friendly* from *friend* and *ly*.

Syntactic

It is knowledge about how the words are organized to construct meaningful and correct sentences.

Pragmatics

It is a high-level knowledge about how to use sentences in different contexts and how the contexts effect the meanings of the sentences.

World

It is useful in understanding the sentence and carry out the conversation. It includes the other person's beliefs and goals.

Speech recognition by machine is important, as many problems get solved if our computer/laptop can recognize the spoken works. We know, in the present days it has become possible to do search in some of the search engines by speaking rather than entering the keywords. In the following, we discuss some of the potential applications of automatic speech recognition (ASR).

Learning Outcomes of this Chapter:

1. Distinguish the goals of sound recognition, speech recognition, and speaker recognition and identify how the raw audio signal will be handled differently in each of these cases. [Familiarity]

2. Implement a feature-extraction algorithm on real data, e.g., an edge or corner detector for images describing a short slice of audio signal. [Usage]
3. Language model and acoustic models for speech recognition. [usage]
4. Implement an algorithm combining features into higher level percepts, e.g., phoneme hypotheses from an audio signal. [Usage]
5. Evaluate the performance of the underlying feature extraction, relative to at least one alternative possible approach (whether implemented or not) in its contribution to the classification task. [Assessment]
6. Tools for speech recognition system. [Usage]

20.2 Automatic Speech Recognition Resources

At the simplest level, the programs that are speech driven can be characterized by the words or phrases we say to a given application and how that application interprets them. An application's active vocabulary is what it listens for, determines what it understands. The speech recognition process makes use of a speech engine, which is language-independent, and what it recognizes can be from several domains. A domain comprises a vocabulary set, pronunciation models, word-usage models that are associated with specific speech applications. An acoustic component is also present in the speech recognition engine, as part of voice models used by the speech engine during the recognition. The voice models can be speaker independent, or may be unique to the speaker.

Figure 20.1 shows the resources used by a typical speech engine during the process of speech recognition. The domain-specific resources can vary dynamically during

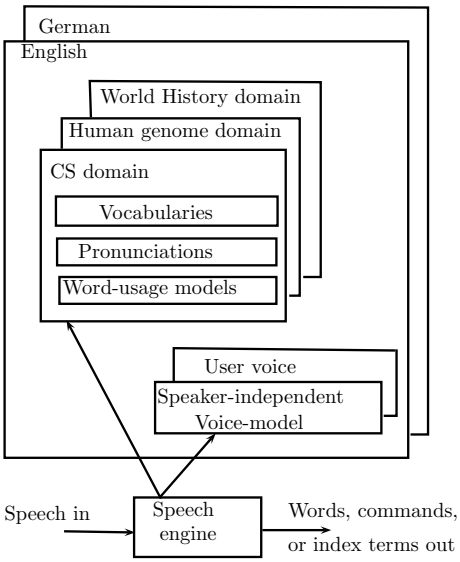

Fig. 20.1 Speech recognition resources

a given recognition session. The vocabulary is one of the domain-specific resources. Some of the major applications are as follows [7].

1. *Dictation application.* It transcribes spoken input directly into the document's text content.
2. *Transaction application.* It facilitates a dialogue leading to a transaction, and
3. *Multimedia indexing application.* This application can generate words, which act as index terms into the multimedia.

As far as application development is concerned, speech engines typically offer a combination of programmable APIs (Application Programming Interfaces) and tools that are helpful to create and define vocabularies and pronunciations for the words they contain. A transactional application may use a smaller, task-specific vocabulary of a few hundred words, but a dictation or multimedia indexing application may use a predefined large vocabulary, some times as large as 100,000 words or so.

A small size of vocabularies is enough for some applications. However, they pose usability problems in the system as their size grows. This is because, the system requires a strict enumeration of the phrases, which must be recognizable by any of the given states in the application. To overcome this limitation, *speech grammars* for specific tasks are defined in the transaction-based applications, which provide an extension to the single words or phrases a vocabulary supports. The speech grammars are helpful in constructing structured collection of words and phrases bound together by rules that define the set of speech streams the speech engine can recognize at a given time. For example, using these grammars, the developers can define a grammar that permits flexible ways of speaking a date, a dollar currency, a number, etc. The prompts that cue users on what they can say next, are an important aspect of defining and using grammar. Further, the speech grammars can serve as a critical component of enabling the voice over the Web, discussed in the next section.

20.3 Voice Web

To have voice facility over the web, a group of industry organizations, which included AT&T, IBM, Lucent, and Motorola, had established the VoiceXML Forum, in March 1999 to develop and promote a new language—the *Voice extensible Markup Language* (http://www.w3.org/Voice/). The VoiceXML forum was established with the objective, to bring the content delivery to interactive voice response applications, using Web-based developments [7, 8].

The VoiceXML (or VXML) is a digital document standard for specifying interactive media and voice dialogues between humans and machines. This language has applications for developing audio and voice response applications, such as for banking systems, customer services, and for automated customer service portals. VoiceXML applications are commonly used in industries and segments of commerce, some of the examples are as follows.

Speech synthesis,
Recognize spoken and touch-tone key input,
Digitize audio,
Order inquiry,
Package tracking,
Flight tracking,
Voice access to email,
Emergency notification,
Audio news magazines,
Can record spoken input,
Like voice dialing, and
Directory assistance.

VoiceXML applications are developed and deployed in all the above major fields. These applications are analogous to how a web browser interprets and visually renders the Hypertext Markup Language (HTML) it receives from a web server. In similar ways, the VoiceXML documents are interpreted by a voice browser, and users interact with voice browsers via the public network like Internet. Like HTML, the VoiceXML text provides *tags* that instruct the voice browser to provide the functions of automatic speech recognition, speech synthesis, dialogue management, and audio playback.

Figure 20.2 shows the architecture of VoiceXML model, which makes use of standard client–server paradigm that integrates voice and data services. A voice service consists of sequence of voice interaction dialogues between a user and an implementation platform. A *document server*, which can be external to the implementation platform, provide the dialogues. The overall service logic is maintained by the document servers or web servers, which also perform database and legacy system operations, and produce dialogues.

A VoiceXML document specifies each interaction dialogue that is conducted by the VoiceXML interpreter. The following is an example of a VoiceXML document:

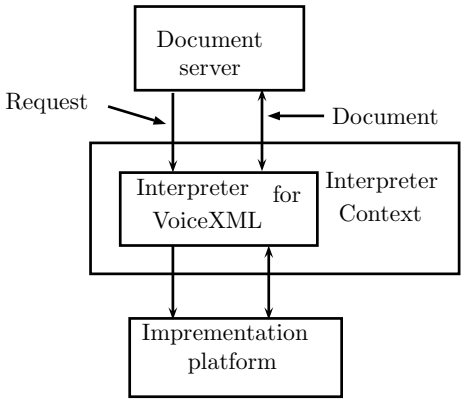

Fig. 20.2 VoiceXML architectural model

```
<vxml version="3.0" xmlns="https://www.w3.org/TR/voicexml30/">
  <form>
    <block>
      <prompt>
        This is a VXML Code!
      </prompt>
    </block>
  </form>
</vxml>
```

A user's input affects dialogue interpretation, and the system collects this information in the form of requests, which it submits to a document server. The later can reply with another VoiceXML document to continue the user's session with other dialogues. The grammar-based recognition vocabularies are commonly used to support voice services in VoiceXML applications.

20.4 Speech Recognition Algorithms

The initial attempts for speech recognition were targeted to use expert knowledge of speech production and perception processes. Soon it was found that such knowledge was inadequate for capturing the complexities of continuous speech. Currently, the statistical modeling techniques trained using hours of speech have provided most speech recognition advancements [4].

The process of speech recognition starts with a sampled speech signal. This signal has a good deal of redundancy because the physical constraints on the articulators that produce speech—the glottis, tongue, lips, and so on—prevent them from moving quickly. Consequently, the ASR (Automatic Speech Recognition) system can compress information by extracting a sequence of acoustic feature vectors from the signal.

Typically, the system extracts a single multidimensional feature vector every 10 ms that consists of 39 parameters. These feature vectors, which contain information about the local frequency content in the speech signal, are called *acoustic observations* because they represent the quantities the ASR system actually observes. The system attempts to infer the spoken word sequence that could have produced the observed acoustic sequence.

To simplify the design, we assume that speaker's vocabulary is known to the ASR system. Having adopted this approach, it is helpful in restricting the search for the possible word sequences only within the words listed in the ASR lexicon. The ASR lists the vocabulary and provides *phonemes*—a set of basic units of words, which are usually individual speech sounds to pronounce each word.

The commercially available lexicons usually include tens of thousands of words. The length of the word sequence uttered by the speaker is not necessarily be known, for the same word by different speakers, as well as by the same speaker at two dif-

ferent times. Consider that length of the word sequence is N. If V is taken as the size of the lexicon, the ASR system can hypothesize V^N possible word sequences. The language constraints dictate that these word sequences are not equally likely to occur. For example, the word sequence "please call me" is more likely to occur than the sequence "please me call." In addition, the acoustic feature vectors extracted from the speech signal can provide important clues about the phonemes which produced them. The sequence of phonemes that corresponds to the acoustics observations, can imply the word sequences that could have produced the sequence of these sounds. Hence, the acoustic observations experienced provide an important source of information that can help further narrow down the space of possible word sequences. The ASR systems use the acoustic observations information to compute the probability that these observed acoustic feature vectors have been produced when the speaker uttered a particular word sequence. Essentially, the system efficiently computes these probabilities and outputs the most probable sequence of words as a *decoded hypothesis*.

The most successful speech recognition systems of today, use a *generative probabilistic model*, shown as Eq. 20.1. The speech recognizer tries to find the probability of word sequence $\hat{w}_1^N$ (of N words) that maximizes the word sequence's probability, having given some observed acoustic sequence y_1^T. This approach makes use of Bayes' theorem to compute the conditional probability of $p(w_1^N)$ given y_1^T. When Bayes equation is expanded in second line of equation (20.1), it ignores the denominator term ($p(y_1^T)$), common for all possible word sequences. This equation maximizes the product of two terms: the probability of the acoustic observations given the word sequence ($p(y_1^T \mid w_1^N)$) and the probability of the word sequence itself ($p(w_1^N)$).

$$\begin{aligned}\hat{w}_1^N &= \underbrace{argmax}_{w_1^N}\ p(w_1^N \mid y_1^T)\\ &= \underbrace{argmax}_{w_1^N}\ p(y_1^T \mid w_1^N)p(w_1^N).\end{aligned} \tag{20.1}$$

Figure 20.3 shows the process described by Eq. 20.1 as a block diagram. The lexicon, language model, and acoustic model components construct hypotheses for

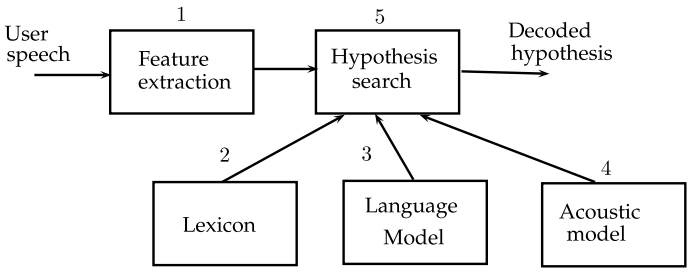

Fig. 20.3 Speech recognition system block diagram

interpreting a speech sample. Block 1, extracts multidimensional features from the sampled speech signal. In Block 5, hypothesis search is carried out, where the search hypothesizes a probable word sequence based on the observation of features, as well the input from three models—lexicon, language, and acoustic. The other components drive the hypothesis search as follows:

- Lexicon in Block 2 defines the possible words that the search can hypothesize, where each word is a linear sequence of phonemes;
- Language model of Block 3 models the linguistic structure (sequence of words i.e., $p(w_1^N)$), but does not contain any knowledge about the relationship between the feature vectors and the words, and
- The acoustic model in Block 4 models the relationship between the feature vectors and the phonemes ($p(y_1^T|w_1^N)$), which might have produced the sounds.

Getting the best performance for feature extraction and hypothesis searches requires customizing the ASR system for individual speakers. The following section explains the hypothesis in detail.

20.5 Hypothesis Search in ASR

Three basic components comprise the hypothesis search: a lexicon, a language model, and an acoustic model. Each one is described in detail in the following [4].

20.5.1 Lexicon

A typical lexicon is shown in Table 20.1 with each lexicon's possible pronunciations constructed from phonemes. English language has 44 phonemes. Despite there being just 26 letters in the language, there are 44 unique sounds (phonemes). These sounds are helpful in distinguishing one word or meaning from another. An individual word can have multiple pronunciations, for example, the word "the" has two pronunciations as shown in Table 20.1. These multiple pronunciations complicate the process of recognition. The hypothesis search chooses the lexicon on the basis of task, trading off vocabulary size with word coverage. Although a search can easily find phonetic representations for commonly used words in various sources, task-dependent jargon often requires writing out pronunciations by hand.

20.5.2 Language Model

The search for the most likely word sequence corresponding to the speech features sampled, requires the computation of terms, $p(y_1^T|w_1^N)$ and $p(w_1^N)$ in Eq. 20.1. The

Table 20.1 Typical lexicon

Lexicon	Phonetic representation
The	dhah
The	dhiy
Cat	kaet
Pig	pihg
Two	tuw

term $p(w_1^N)$ is called the *language model*. The computation requires the assignment of probability to a sequence of words w_1^N. A simplest way we can imagine to determine such a probability, is to compute the relative frequencies of different word sequences, like we discussed earlier, that "Please call me" is more probable than "Please me call." Note that, the total number of different sequences can grow exponentially with the length of the sequence, making this approach computationally infeasible. Therefore, there is a need of approximations.

A typical approximation used assumes the probability of current word in the sequence as depending on previous two words only, called 2-grams, against n-grams. When this is considered, the computation can approximate the probability of the word sequence as follows:

$$p(w_1^N) \approx p(w_1)p(w_2|w_1)\prod_{i=3}^{i=N} p(w_i|w_{i-1}, w_{i-2}). \tag{20.2}$$

The term $p(w_i|w_{i-1}, w_{i-2})$ can be estimated through computation by counting the relative frequencies of word *trigrams*, or triplets:

$$p(w_i|w_{i-1}, w_{i-2}) \approx \frac{N(w_i, w_{i-1}, w_{i-2})}{N(w_{i-1}, w_{i-2})}. \tag{20.3}$$

In the above, N is the associated event's relative frequency. Typically, training such a language model requires using hundreds of millions of words to estimate $p(w_i|w_{i-1}, w_{i-2})$ for different word sequences. Even then, many trigrams do not occur in the training text, so the computation must smooth the probability estimates to avoid zeros in the probability assignment.

20.5.3 Acoustic Models

An acoustic model computes the probability of feature vector sequences (y_1^T) under the assumption that a particular word sequence (w_1^N) produced the vectors. In other words, an acoustic model is $P(y_1^T|w_1^N)$.

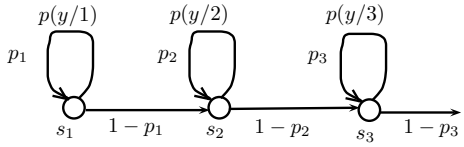

Fig. 20.4 Hidden Markov model for a phoneme

Due to inherently stochastic nature of the speech, a speaker never utters a word exactly the same way twice. The variation in a word's or phoneme's pronunciation manifests itself in two ways: duration and *spectral contents*, also known as *acoustic observations*. In addition, a particular phoneme's spectral content are effected due to phonemes in surrounding context, a phenomenon called *co-articulation* effect. It is, therefore, necessary that acoustic models used should take into account these co-articulation effects. One of the popular acoustic models is based on HMM (Hidden Markov Model).

Hidden Markov Models

A hidden Markov model offers a natural choice for modeling speech's stochastic aspects. HMMs function as probabilistic finite-state machines—the model has a set of states, and its topology specifies the allowed transitions between them. At every time frame, an HMM makes a probabilistic transition from one state to another and emits a feature vector with each transition.

Figure 20.4 represents a *phoneme*'s transitions using a HMM. The transitions in the waveform of a phoneme correspond to state transitions in the HMM. We may think of a HMM as a finite automata with transitions governed by probabilities. Accordingly, we take a set of state transition probabilities in the HMM as, p_1, p_2, and p_3, due to which the possible transitions between the states of the HMM are governed. The probabilities specify the probabilities of going from one state at time t to another state at time $t+1$. The feature vectors emitted while making a particular transition in the speech waveform, represent the spectral characteristics of the speech at that point. These feature vectors vary corresponding to varying pronunciations of the phoneme. A probability distribution or probability density function can model this variation. In Fig. 20.4, the functions $p(y|1)$, $p(y|2)$, $p(y|3)$, could be different for different transitions. Typically, these distributions are modeled as parametric distributions, which are a mixture of multidimensional Gaussian distributions.

The HMM in Fig. 20.4 has three states, representing the pronunciation of a phoneme starting at state s_1. Then, the phoneme corresponds to a sequence of transitions, and terminating at state s_3. Duration of a phoneme is equal to the number of time frames required to complete the transition sequence. The transition probabilities $p_1 \ldots p_3$ implicitly indicate probability distribution that governs the duration of the phoneme. If any of these transitions exhibits high *self-loop*[1] probabilities, the model spends more time in that state, consequently consuming a longer time

[1] for example, the word "speech" can be also pronounced as "spee...ech", repeating the sound of 'e', which creates a self-loop.

to go from the first to the third state. Note that, how time duration a self-loop may repeat, in unknown and varies from speaker to speaker. However, a self-loop (a state) may repeat on itself few times, typically 2-5. However, some words' phoneme(s) may have exceptionally long loop, for example, chanting of *Aum*.[2] The probability density functions associated with these three transitions govern the feature vector output.

A fundamental task required to be performed through an HMM is computation of the likelihood that it produces a given sequence of acoustic feature vectors. For example, assume that the system extracted T *feature vectors* from speech corresponding to the pronunciation of a single phoneme, now the system seeks to infer which phoneme from a set of, say, 50 was spoken, given these feature vectors. The procedure for inferring the phoneme first assumes that the ith phoneme was spoken, then finds the likelihood that the HMM for this phoneme produced the observed feature vectors. The system then hypothesizes that the spoken phoneme model is the one which has the highest likelihood of matching the observed sequence of feature vectors.

If the sequence of HMM states is known, we can easily compute the probability of a set of feature vectors. For this, the system computes the likelihood of the tth feature vector y_t using the probability density function for the HMM state at time t. Having done this, the likelihood that set of all the T feature vectors has occurred is simply the product of all the individual likelihoods y_t. Usually, it is not possible to know the actual sequence of state transitions, for the computation of likelihood, all possible state sequences are summed. Given that the HMM dependencies are local, it is possible to derive efficient formulas for performing these calculations recursively.

Parameter Estimation

It is necessary that in advance to using of an HMMs to compute the likelihood values of feature vector sequences, the HMMs needs to be trained to estimate the model's parameters. The training process requires the availability of a large volume of training data in the form of mappings of "spoken word sequences" versus "feature vectors" extracted from the corresponding speech signals. The commonly used process to find a particular correspondence is, estimation of *maximum likelihood* (ml) function ($\hat{\theta}_{ml}$). Given that a correct word sequence is known corresponding to the feature vector sequence, the maximum likelihood computation process tries to choose those HMM parameters, which maximize the likelihood of training feature vector. The computation of feature vectors also keeps target for obtaining the correct word sequence. Consider that y_1^T is the representation for stream of T acoustic observations, and let w_1^N represents the correct word sequence; for these, the maximum likelihood function estimate represented by $\hat{\theta}_{ml}$ is,

$$\hat{\theta}_{ml} = \underbrace{argmax}_{\theta} \ \log[p_\theta(y_1^T|w_1^N)]. \tag{20.4}$$

[2]chanting of *Aum* is a common practice during meditation and yoga (IPA:/ɐwm/), where sound of IPA ' 'm' is repeated.

In the beginning, an HMM is constructed for a correct word sequence to start the training process. Then, for each next word, the HMM is constructed by concatenating the HMMs for the phonemes that constitute the next word. The word HMMs are concatenated to construct the HMM for the complete utterance. As an example, the words "we" and "were" have corresponding phonemes as "W IY" and "W ER", respectively. Hence, HMM for the utterance "we were" would consist of the concatenation of HMMs of four phonemes "W IY W ER".

The training phase of an HMM assumes that it is possible to obtain the acoustic observations y_1^T by the system, by traversing HMM from initial state to final state in total T time-frames. However, we know that system cannot trace the actual state sequence, e.g., due to the loops. Therefore, *ml* estimation assumes that this state sequence is hidden, and thus, what is best possible is to average out all the state sequence values. The system can express the maximization of Eq. 20.4 in terms of the HMM's hidden states s_t at time t, as follows:

$$\underbrace{argmax}_{\theta} \sum_{t=1}^{T} \sum_{s_t} p_\theta(s_t|y_1^T) \log[p_{\hat{\theta}}(y_t|s_t)]. \tag{20.5}$$

An iterative process is used to solve Eq. 20.5, with each iteration having two steps: 1. An expectation step, and 2. A maximization step. The expectation step computes the posterior probability $p_\theta(s_t|y_1^T)$, or count of a state. The posterior probability is conditioned on all the acoustic observations. The system makes use of current parameter estimate of HMM, and the computation is performed using forward–backward algorithm. The parameter $\hat{\theta}$ is chosen by the maximization step to maximize Eq. 20.5. For Gaussian nature of the probability function, the computation can derive closed-form expressions for this step [4].

20.6 Automatic Speech Recognition Tools

Before we proceed to understand some of the commonly available tools for speech recognition, let us try to understand some of the important terminology of speech recognition.

Automatic speech recognition (ASR) or simply the speech recognition, or computer-based speech recognition, is a process of converting the speech into a sequence of words, using some algorithms which have been implemented as programs. A standard way of doing this is to first split the utterances in the speech waveform with respect to certain parameters of speech recognition. Some of these are: presence of voice activity, duration, pitch, voice quality, voice intensity, S/N (signal to noise) ratio, and the strength of *Lombard effect*.[3] This is followed with

[3] *Lombard effect*: Involuntary tendency of speakers to increase their vocal effort particularly when speaking in loud background to enhance the audibility of their voice. Due to the Lombard effect, not

recognition of each utterance. The important factors to be considered during the recognition are detailed in the following paragraphs.

Concept of Features

The parameters or values in a complete waveform signal for a given speech is very large. Therefore, to optimize it, a given speech is divided into a large number of *frames*, such that each frame is of small length, typically 10 milliseconds. Each such frame is used to extract 39 different *features*, i.e., 39 different numerical values representing speech optimized to 1 frame. These values of one frame are called *feature vector*. Each such vector that corresponds to a speech segment of ten milliseconds is numerical representation of the speech of that duration.

Concept of Model

We need a mathematical model, called *concept model*, that has a representation to gather common attributes of spoken words. Such a model needs to be evaluated for various characteristics, like, how adaptive it is for the changing situations, how well this model fits into practice, and how well it can be configured?

Concept of Matching Process

It would require a lot of time for comparing the feature vectors with all the models. Hence, the search should be optimized for choosing the best matching variant.

20.6.1 Automatic Speech Recognition Engine

Julius is a high- performance continuous speech recognition software, based on word N-grams. It is able to perform recognition at the sentence level with a vocabulary of the order of tens of thousands of words. Julius can realize high-speed speech recognition on ordinary laptop/PC, can perform the speech recognition at near real time, and can achieve typically a recognition rate of greater than 90% using a vocabulary of 20, 000-words, for dictation tasks. Julius is multipurpose, i.e., by recombining the pronunciation dictionary, language, and acoustic models, one can build task-specific systems. Its code is open source, so one can recompile the system for other platforms or alter the code for specific needs [2].

Figure 20.5 shows the structure of *Julius* speech recognition system. The language model of Julius uses N-gram mode, and context-dependent HMM (Hidden Markov Model) is used as Acoustic model. As shown in the figure, the input speech is processed through two passes: first pass is 2-gram frame synchronous beam-search (a high-speed approximate search), and second pass is 3-gram N-best stack decoding, which is a high precision technique. It can do online recognition using PC/laptop's

---

only the loudness increases, but also the other acoustic features such as pitch, rate, and duration of syllables. The Lombard effect also results in an increase in the signal-to-noise ratio of the speaker's signal.

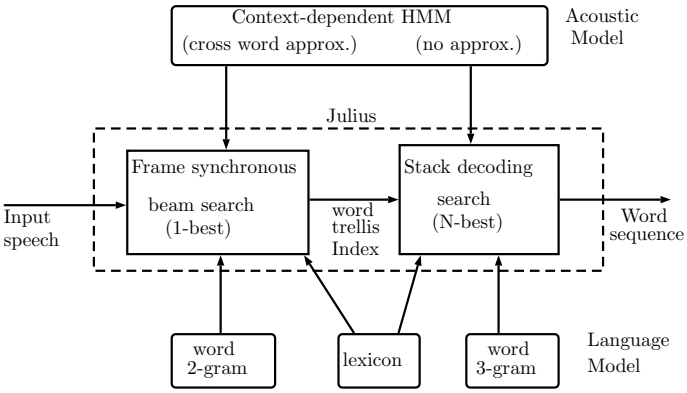

Fig. 20.5 Julius speech recognition system

microphone or can use any audio device. The first pass of Julius segments input with short pauses, and the second pass sequentially decodes these segments and slots to the results. During the first pass, when a short pause has the maximum likelihood at a certain point of time, a break is placed at that point and second pass is executed on that utterance segment. Due to this process, word constraints are preserved as the context within a utterance segment, and first pass may continue over to the next utterance. Using the above sequence of steps, an input speech file with multiple sentences can be decoded.

20.6.2 Tools for ASR

Speech Recognition is available in English, and many other languages, and this feature is common in most smart phones, laptops, and PCs. In the following, we discuss basic tools available as open source.

Kaldi Speech Recognition Toolkit

Kaldi is an open-source speech recognition toolkit, written in C++ language, and works under the Apache platform.

It is a finite-state transducer (FST) based framework, with linear algebra support. Figure 20.6 shows different components of Kaldi: the library modules are grouped together, which depend on two types of libraries, 1. linear algebra libraries (the numerical algebra libraries) and 2. OpenFST (finite-state framework). These two external libraries are also freely available as open source [3, 5].

Access to the library functionalities is provided through command-line tools written in C++. These tools are called from a scripting language, for building and running a speech recognizer. Each tool has a very specific functionality with a small set of command-line arguments: for example, there are separate commands to be executed

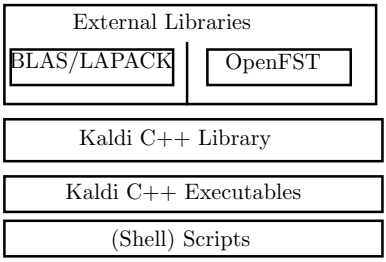

Fig. 20.6 Kaldi speech recognition toolkit

for accumulating statistics, summing accumulators, and updating a GMM-based (Gaussian Mixture Models) acoustic model. For language modeling (LM), Kaldi uses FST-based framework, hence, in principle it can use any language model that can be represented as FST.

CMU-Sphinx

Since its release as an open-source code in 1999, CMU-Sphinx provides a platform for building speech recognition applications. It is used in desktop control software, telephony platforms, intelligent houses, computer-assisted language learning tools, information retrieval, and mobile applications. Traditionally, CMU-Sphinx provides support for low-resource and underdeveloped languages. It is a speech recognition toolkit with tools to build speech applications, which makes use of technologies such as C, cross-platform, HMM (Hidden Markov Models), JavaScript, and Python. CMU-Sphinx contains a number of packages for different tasks and applications. The following is the list:

Sphinx4—adjustable, modifiable recognizer written in Java, and
Sphinxtrain—acoustic model training tools.
Pocketsphinx—lightweight recognizer library written in C, and
Sphinxbase—support library required by Pocketsphinx.

Deep Speech Tool

It is a simple end-to-end deep learning based speech system, which when combined with a language model, achieves higher performance than traditional methods on hard speech recognition tasks. The deep tool is realized by training a large recurrent neural network (RNN) that uses multiple GPUs and thousands of hours of data. Due to which the system learns directly from data, and there is no need for specialized components for *speaker adaption*, like in other systems, neither it needs noise filtering. The more traditional systems use acoustic models and Hidden Markov Models(HMM) [1].

The RNN, which is core of this system, is trained to speech spectrograms to generate English text transcriptions. A training set $\chi = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ...\}$ is used to sample a single utterance x and a label y, where each utterance $x^{(i)}$ is a time series of length $T^{(i)}$. Here every time slice is a vector of audio features, $x_t^{(i)}, t = 1, ..., T^{(i)}$. The role of RNN is to convert the input sequence x into a sequence of character probabilities for the transcription y, with $\hat{y}_t = p(c_t|x)$, where $c_t \in \{a, b, c, ..., x, space, \prime\}$.

The RNN has five layers of hidden units, such that for an input x, the hidden units at layer l are denoted as $h^{(l)}$, with input as $h^{(0)}$. The first three layers are not kept as recurrent layers. At each time t, for the first layer, the output depends on the spectrogram frame x_t along with a context of C frames on each side. The remaining non-recurrent layers operate on independent data for each time step. Thus, for each time t, the first three layers are computed by

$$h_t^{(l)} = g(W^{(l)} h_t^{(l-1)} + b^{(l)}), \tag{20.6}$$

where $g(z) = min\{max\{0, z\}, 20\}$ is the clipped rectified-linear activation function and $W^{(l)}$, $b(l)$ are weight matrix and bias parameters for layer l.

HTK Tool

The hidden Markov model toolKit (HTK) is a portable toolkit for building and manipulating hidden Markov models. It is primarily used for *speech recognition* research, and *speech synthesis*, as well for character recognition and DNA sequencing. HTK consists of set of library modules and tools available in ANSI C source form. These tools provide sophisticated facilities for speech analysis, HMM training, testing, and result analysis.

The statistical speech models use here the context-dependent hidden Markov models. Probabilities of word sequences is based on N-gram, which finds the most probable word sequence using language model (refer Eqs. 20.1, 20.2) and acoustic model (refer Eq. 20.1).

20.7 Summary

Automatic speech recognition (ASR) is another domain of human–machine interface, where machine is to recognize human speech, that is, transform some kind of frequency signal to text. This is a complex process, and requires many processes like phonological, morphological, syntactic, pragmatics, and world. Speech recognition is considered as the future of computer interface. There are basically two application modes for speech recognition: 1. Using speech as input, or 2. As data or knowledge. The application of "speech as input" addresses applications like dictation systems, navigation or transaction systems (like purchasing stocks). Using speech as knowledge has applications like meeting capture.

An application of speech recognition can be implemented using "Voice extensible Markup Language" (http://www.w3.org/Voice/). Developers can use *VoiceXML* to create audio dialogues that feature synthesized speech, recognition of spoken and touch-tone key input, digitized audio, recording of spoken input, telephony, and mixed-initiative conversations. The VoiceXML's architecture uses client–server paradigm to integrate voice services with data services. A voice service is a sequence

of interaction dialogues between a user and an implementation platform, and a document server, which can be external to the implementation platform, and can provide the dialogues.

The modern speech recognition algorithms are based on statistical modeling techniques trained from hours of speech. The process of speech recognition starts with a sampled speech signal, which has a good deal of redundancy due to the physical constraints on the articulators that produce speech. Consequently, the ASR system can compress information by extracting a sequence of acoustic feature vectors from the signal. A system extracts a single multidimensional feature vector every 10 ms that consists of 39 parameters. The system seeks to infer the spoken word sequence that could have produced the observed acoustic sequence, using Bayesian probabilistic approach, which basically, is a process of hypothesis search. Models used for AR are: Language model (searching the most likely word sequence), Acoustic models (compute the probability of feature vector components), and Hidden Markov models (probabilistic finite-state machines).

There are a number of software tools, most as open source, for speech recognition, speech synthesis, as well for research in automatic speech recognition. These software tools take input as sound-wave signal (a file) and split the waveform based on the utterances by the speaker, sample the speech input intervals of about 10 milliseconds, extract the features of input speech. Then using the various models of probability theory, estimate the probable text, which most likely would have produced these utterances. These tools were developed (mostly) as research projects. Among these are: Julius, Kaldi, CMU-Sphinx, Deep Speech tools, and HTK (Hidden Markov Model).

Exercises

1. Consider alphabet set $\Sigma = \{a, b, c, d\}$. Create finite automata (recognizers) for following strings.

   a. All strings which start with letter a.
   b. All strings which end with letter d.
   c. All strings where every c is followed letter d.
   d. All strings which have odd number of c's.

2. Answer followings in brief, giving suitable examples.

   a. What is the difference between *phoneme* and *morpheme*?
   b. What is the difference between *language* and *dialect*?

3. Write an equation to compute trigram probability.

4. The text processing algorithms are usually written in Python, while the ASR algorithms, which produce the same text, are written in C/C++. Explain what could have been the reason behind this?
5. What is the fundamental difference between the *language* model and *acoustic* model? Why are they the same so?

References

1. Hannun A et al (2014) Deep Speech: Scaling up end-to-end speech recognition. https://arxiv.org/abs/1412.5567. Accessed Dec 19, 2017
2. http://julius.osdn.jp/book/Julius-3.2-book-e.pdf. Accessed Dec 19, 2017
3. http://kaldi.sf.net/. Accessed Dec 19, 2017
4. Padmanabham M, Picheny M (2002) Large-vocabulary speech recognition algorithms. Computer 4:42–50
5. Provey D (2011) The Kaldi speech recognition toolkit. IEEE workshop on automatic speech recognition and understanding. US IEEE Signal Processing Society, Hawaii
6. Ronald C et al (1997) Survey of the state of art in human language technology. Studies in Natural Language Processing, Cambridge University Press
7. Savitha S, Eric B (2002) Is speech recognition becoming mainstream? Computer 4:38–41
8. http://www.w3.org/Voice/. Accessed Dec 19, 2017