The Julius book

Akinobu LEE

May 17, 2010

The Julius book
by Akinobu LEE

Edition 1.0.3 - rev.4.1.5

Contents

Preface

"Julius" is an open-source, high-performance large vocabulary continuous speech recognition (LVCSR) decoder software for speech-related researchers and developers. Based on word N-gram and triphone context-dependent HMM, it can perform almost real-time decoding on most current PCs with small memory footprint.

It also has high vesatility. The acoustic models and language models are pluggable, and you can build various types of speech recognition system by building your own models and modules to be suitable for your task. It also adopts standard formats to cope with other toolkit such as HTK, CMU-Cam SLM toolkit, etc.

The core engine is implemented as embeddable library, to aim to offer speech recognition capability to various applications. The recent version supports plug-in capability so that the engine can be extended by user.

Julius is an open-source software, and is available for free with source codes. You can use Julius for any purpose, including commercial ones, at your own risk. See the license document as included in the package for details.

Our motivation to develop such an open-source speech recognition engine is to promote the high-standard recent advances in speech recognition studies toward open community, and to encourage speech processing related researches and developments on various fields. The first version of Julius was released on 1996, and due to its technical challenges and public needs, this work still continues until now.

Julius is being maintained at the institutes and groups listed below. To make a contact, please E-mail to julius-info at lists.sourceforge.jp, or access directly to the developer or maintainer.

The project Web page is located at http://julius.sourceforge.jp . You can get the latest version of Julius, several models, documents and source code references. You can also obtain the source snapshot of the current development version via CVS. There is also a web forum for the developers and users using Julius.

The "Julius" was named after "Gaius Julius Caesar", who was a "dictator" of the Roman Republic in 100 B.C.

This is a total reference book of Julius.

Chapter 1

Overview

This chapter describes the general information of Julius. The system requirements, models and package overview is described.

1.1 System Requirement

Julius is developed under Linux and Windows. It can also run on many Unixens like Solaris, FreeBSD and MacOS X. Since Julius is written in pure C and has little dependency on external libraries, it can run on other platforms. Developers has been ported Julius to Windows Mobile, iPhone and other microprocessor environments.

Julius supports a recognition of live speech input via audio capture device at all the supported OS above. See the "Audio Input" Chapter for the list of requirements for live input on each OS.

1.2 Things needed to run speech recognition

To perform speech recognition with Julius, you should prepare "models" for the target language and task. The models should define the linguistic property of the target language: recognition unit, audio properties of each unit and the linguistic constraint for the connection between the units. Typically the unit should be a word, and you should give Julius these models below:

- "Word dictionary", which defines vocabulary. It deines the words to be recognized and their pronunciations as a phoneme sequence.
- "Language model", which defines syntax level rules that defines the connection constraint between words. It should give the constraint for the acceptable or preferable sentence patterns. It can be eigher a rule-based grammar, or probabilistic model such as word N-gram. The language model is not needed for isolated word recognition.
- "Acoustic model", which is a stocastic model of input waveform patterns, typically per phoneme. Julius adopts Hidden Markov Model (HMM) for the acousic modeling.

Since Julius itself is language-independent decoding program, it can run for a new language if given dictionary, language model and acoustic model for the language.

Julius is a mere speech decoder which computes most likely sentence for given input, so the recognition accuracy largely depends on the models.

Julius adopts acoustic models in HTK ascii format, pronunciation dictionary in almost HTK format, and word 3-gram language models in ARPA standard format (forward 2-gram and reverse 3-gram trained from same corpus).

You can get standard Japanese models for free from the Julius web site, and more various models is being delivered at Continuous Speech Recognition Consortium, Japan. For more detail, please contact csrc@astem.or.jp.

For English, we currently have a sample English acoustic model trained from the WSJ database. According to the license of the database, this model CANNOT be used to develop or test products for commercialization, nor can they use it in any commercial product or for any commercial purpose. Also, the performance is not so good. Please contact to us for further information.

More up-to-date information can be obtained on the Web page.

1.3 Tools and libraries in the distribution

Julius is distributed basically as source archive, and binary packages for Linux and Windows are also available. The source archive contains full program sources of Julius and related tools, release information, sample configuration file, sample plugin source codes and Unix online manuals. The binary packages are based on the source archive, containing pre-compiled executables and related files extracted from the source archive. You can also get a development snapshot via CVS.

These tools are included:

- julius --- main speech recognition software Julius
- adinrec --- audio detection/recording check tool
- adintool --- a tool to record / split / send / receive audio streams
- jcontrol --- a sample module client written in C
- jclient.pl --- a sample module client written in Perl
- mkbingram --- convert ARPA N-gram file into binary format
- mkbinhmm --- convert HTK ASCII hmmdefs file into binary format
- mkbinhmmlist --- convert HMMList file into binary format
- mkgshmm --- convert monophone HMM to GS HMM for Julius
- mkss --- calculate average spectrum for spectral subtraction
- Tools for language modeling --- mkdfa.pl, mkfa, dfa_determinize, dfa_minimize, accept_check, nextword, generate, generate-ngram, gram2sapixml.pl, yomi2voca.pl

In Linux, the libraries, header files and some scripts will be installed for development.

- `libsent.a` --- Julius low-level library
- `libjulius.a` --- Julius main library
- `include/sent/*` --- headers for libsent
- `include/julius/*` --- headers for libjulius
- `libsent-config`, `libjulius-config` --- scripts to get required flags for compilation with libsent and libjulius

Chapter 2

Installation

This chapter describes how to compile and install Julius. Julius can be run without system installation, but system installation is recommended when you are using Julius library for a software development.

The compilation procedure from source archive is fully described in this chapter. All the compilation-time options and OS-specific matters are also explained here.

2.1 Install from binary package

All the executables are located at the `bin` directory under the package root directory. Since Julius itself is a stand-alone application, they can be run directly without installation. However, you can install it to your system by manually copying `bin`, `include`, `lib` and manuals to the corresponding system directory. For example, if you want to install Julius files under `/usr/local`, the copy source and destination will be:

```
bin/* -> /usr/local/bin
include/* -> /usr/local/include
lib/* -> /usr/local/lib
doc/man/man1/* -> /usr/local/man/man1
doc/man/man5/* -> /usr/local/man/man5
```

2.2 Compile from source

When you want to change some sompile-time settings of Julius (ex. vocabulary size limit or input length limit, search algorithm variants. ...). you should compile Julius from the source codes. Windows (MinGW and Cygwin), Linux, Solaris and other unixen are supported to use the automatic configuration. The latest Julius also has support for Microsoft Visual C++.

Julius adopts autoconf scheme to be easily compiled on various Unix-like environments. Just go to the directory where you unpacked the source archive, and do the following commands,

```
% ./configure
% make
```

and install the generated binary files by

```
% make install
```

which also installes headers and libraries as well as the binaries.

You can tell the **configure** script to use another compiler by setting environment variable `CC`. The compliation flags can be also specified by `CFLAGS`. Here is an example to specify which compiler and flags to be used on bash-based shell.

```
% export CC=cc
% export CFLAGS=-O3
% ./configure
```

At the last step, the "make install" commadn will copy executables, libraries, headers and manuals to the specified location of the system. The target installation directory is listed below, in which `${prefix}` is a prefix of system directory, which typically will be set to `/usr/local` by default but can be altered by **configure** option `--prefix=...`.

```
bin -> ${prefix}/bin
include -> ${prefix}/include
lib -> ${prefix}/lib
doc/man/man1 -> ${prefix}/man/man1
doc/man/man5 -> ${prefix}man/man5
```

For example, if you want to install Julius at the "`$HOME/julius`", the options should be like this:

```
% ./configure --prefix=$HOME/julius
```

Julius has many configure options other than standard ones to set application-specific parameters. See the next section for details.

2.3 Configuration options

This section describes **configure** options that can be given to the configure script at the first step of compilation. Here options are grouped into three groups corresponding to the subdirectories of the source archive: `libsent` options, `libjulius` options and `julius` options. You can give all the options together at one time to the configure script of the top directory (in that case all the options are passed to the configure script of the subdirectories and irrelevant ones are omitted), or give them separately to the corresponding configure script.

2.3.1 libsent options

The "libsent" library, located on the `libsent` directory, contains a collection of general functions that are required for a speech recognition system: audio I/O, network I/O, preprocessing, speech feature extraction, language model, acoustic model, output probability computations, transition handling, indexing and so on.

`--enable-words-int` By default the maximum vocabulary size is 65,535. This limit comes from the fact that the internal word ID type is defined as `unsigned short`. This option will tell Julius to assign the word ID as `int`, wihch will extend the size limit at a cost of increasing memory usage.

`--enable-msd` Enable MSD-HMM support on acoustic modeling.

`--disable-class-ngram` Disable class N-gram support. This will save memory for less than 100 kBytes if you don't use class N-gram.

`--enable-fork` With this option, Julius will fork at each audio client connection (`-input adinnet`).

`--with-mictype={auto|oss|alsa|esd|portaudio|sp|freebsd|sol2|sun4|irix}` Specify A/D-in device for microphone input. When `auto` is specified (this is default), it tries to find an available one automatically. On Linux system, `alsa`, `oss`, `esd` will be examined by this order and first one will be chosen as default. On Windows, `portaudio` will be chosen when the required DirectSound headers are available, otherwise WMM driver will be used.

`--with-netaudio-dir=dir` For DatLink users, specify this option to enable direct input from NetAudio server on DatLink. The `dir` should be the directory where the NetAudio SDK (`include` and `lib`) is located.

`--disable-zlib` Disable linking to zlib library. A compressed file may not be read if you use this option.

`--without-sndfile` Disable using libsndfile for audio file reading.

2.3.2 libjulius options

The `libjulius` is the core recognition engine library performing actual recognition.

`--enable-gmm-vad` This option enables GMM-based front-end VAD on Julius. See the "voice activity detection" section of this document for more details.

`--enable-decoder-vad` This option enables decoder-based VAD on Julius. The enabled feature will be activated with `-spsegment` option at run time. See the sections of "voice activity detection" and "search algorithm" for more details.

`--enable-power-reject` Enables input energy based input rejection. See sections of "voice activity detection" and "input rejection" in this document.

`--enable-setup={standard|fast|v2.1}` Configure the detailed search algorithms by specifying one of the preset values:

- `fast`: speed-tuned preset (default)
- `standard`: accuracy-tuned preset
- `v2.1`: old preset, compatible with ver.2.1.

`fast` tells Julius to use faster algorithm using several approximations on score compuration and aggressive pruning on search. It runs faster than "standard", but these approximations and prunings may result in a slight degradation of recognition accuracy of several percent. So, it may be better to use another option if you are going to evaluate an acoustic model or a language model through recognition result.

`standard` make Julius to do accurate recognition with minimum (ideally no) loss of accuracy by engine. It has less accuracy loss as compared with `fast` setting, but it will take longer decoding time.

`v2.1` is an old option that reverts all the algorithm equivalent to the ancient version 2.1.

The detailed set of algorithms that will be enabled or disabled by this option is summarized at the table below. The "argument" row corresponds to each option described in the following.

```
           |1-gram     1st pass 2nd pass    tree       Gauss. pruning
           |factoring  IWCD     strict IWCD separation default method
===========+=============================================================
argument   | factor1    iwcd1   strict-iwcd2  lowmem2
-----------+-------------------------------------------------------------
 standard  |  o          o         o           x        safe
   fast    |  o          o         x           o        beam
   v2.1    |  x          x         x           x        safe
-------------------------------------------------------------------------
```

`--enable-factor2` Use 2-gram factoring on the 1st pass. By default Julius uses 1-gram factoring. Enabling this option will improve the accuracy of the 1st pass, but perhaps has little effect on the final output. It also costs much time and memory.

`--enable-wpair` Use word-pair approximation instead of 1-best approximation on the 1st pass. The word-pair approximation improves the accuracy of the 1st pass by treating independent hypotheses for all predecessor words, but it costs much memory.

`--enable-wpair-nlimit` When specified with `--enable-wpair`, this option will limit the number of independent predecessors per word.

`--enable-word-graph` Internally generate word graph instead of word trellis at the end of the 1st pass.

`--disable-pthread` Disable creating separate thread for audio capturing. May be required on OS which does not support threading.

`--disable-plugin` Disable plugin feature.

2.3.3 julius options

The directory `julius` contains the main function of the software "julius". It links to the `libsent` and `libjulius` libraries to build a stand-alone, server-client speech recognition "julius". The application utilities such as character set conversion are implemented here.

`--enable-charconv={auto|iconv|win|libjcode|no}` Specify which scheme to use for charset conversion. Choosing `iconv` will tell Julius to use the iconv library, and `win` to use the native API of Windows. `no` will entirely remove charset conversion feature. Default value is `auto`, in which `iconv` will be chosen on Linux and `win` on Windows.

2.4 Building Julius on various platform

This section describes the detailed procedure and requirements to to compile Julius on various platforms.

2.4.1 Linux

These libraries are required to build Julius.

- `zlib`
- `flex`

These packages are recommended to be installed before compilation of Julius, although Julius can be compiled without these files.

- ALSA headers and libraries. If not exist, Julius uses OSS interface by default
- ESounD headers and libraries for live audio input via esd daemon
- `libsndfile` libraries for audio file reading. If not, only .wav and .raw files can be read.

Many linux-based system offers package management system to install these packages. For example, on a Debian-based distribution, you can install all the required / recommended packages by executing the following command at root (the package names may vary on each distribution).

```
# aptitude install build-essential zlib1g-dev flex
# aptitude install libasound2-dev libesd0-dev libsndfile1-dev
```

2.4.2 Windows - cygwin

The following packages should be installed tbefore compiling Julius:

```
Devel
    - binutils
    - flex
    - gcc-core
    - gcc-mingw-core
    - libiconv
    - make
    - zlib-devel
Utils
    - diffutils
Perl
    - perl
MinGW
    - mingw-zlib
```

Moreover, these DirectSound SDK header files are needed compile Julius with live audio recognition support:

```
d3dtypes.h
ddraw.h
dinput.h
directx.h
dsound.h
```

These DirectSound development files can be found in the Microsoft DirectX SDK. If the SDK is installed on your machine, you can find the DirectSound headers somewhere in the SDK. You should find them, and copy them to the directory to `/usr/include` and `/usr/include/mingw` in the cygwin environment before executing the **configure** script.

The actual building procedure is as the same as Linux. If you want to build .exe files that can run outside cygwin, you should give the gcc an option `-mno-cygwin`. You can tell **configure** to use the option at any time by:

```
% CC="gcc -mno-cygwin" configure
```

When Julius fails to find the DirectSound headers, it will be compiled with old API called "mmlib" You can check `julius.exe` whether the compiled binary uses DirectSound or not by executing the command below. If it is compiled with DirectSound, the line should contains the "pa-dsound" string like this:

```
% julius.exe --version
...
primary A/D-in driver: pa-dsound (PortAudio ....)
...
```

2.4.3 Windows - mingw

Julius can be compiled at MinGW (Minimalist GNU for Windows) environment. MinGW easily allow users to build binaries that can be run without MinGW, so it is suitable to build a dirtibutable binary.

You should install MinGW, MSYS and msys-DTK to compile Julius. Additionally, Win32 library of "zlib" and "flex", and DirectSound headers are required. The zlib, flex libraries and DirectSound headers are not included in the standard mingw distribution, so you have to obtain them by your own and install them to the system directories, ex. `/mingw/lib/` and `/mingw/include/`. Actually, these files below are needed.

```
/mingw/include/d3dtypes.h
/mingw/include/ddraw.h
/mingw/include/dinput.h
/mingw/include/directx.h
/mingw/include/dsound.h
/mingw/include/zconf.h
/mingw/include/zlib.h
/mingw/lib/libfl.a
/mingw/lib/libz.a
```

The compilation and installation procedure is just as same as Linux.

```
% ./configure
% make
% make install
```

2.4.4 Windows - Microsoft Visual C++

Julius-4.1.3 and later now supports compilation on Microsoft Visual C++ (MSVC). In addition to the command-based executables, an additional sample application named "`SampleApp`" is included to demonstrate the C++ wrapper implementation of Julius.

The MSVC support has been tested on MS Visual C++ 2008, both Professional and Express Edition.

2.4.4.1 Requirements

"Microsoft DirectX SDK" is required to compile Julius on MSVC. You can get it from the Microsoft Web site. Install it before compilation.

Also, Julius uses these two open-source libraries:

- zlib
- portaudio V19

They are already included in the Julius source archive at "`msvc/zlib`" and "`msvc/portaudio`" so you need not prepare by yourself.

2.4.4.2 Build

Open the solution file "`JuliusLib.sln`" with MSVC, and build the projects in the following order:

- `libsent`
- `libjulius`
- `julius`
- `SampleApp`

After the build process, you will get the Julius libraries and executables "`julius.exe`" and "`SampleApp.exe`" under "`Debug`" or "`Release`" directory.

If you got an error when linking zlib or portaudio, compile them by yourself and replace the headers and libraries under each directory. When you compiled the portaudio library, you also have to copy the generated DLL ("`portaudio_x86.dll`") under the "Release" and "Debug" directories.

2.4.4.3 Testing the sample application

"`julius.exe`" is a win32 console application, which runs as the same as other Unix versions. You can run it from command prompt with a working jconf file just like Linux versions.

The "`SampleApp.exe`" is a C++ sample application which defines a simple Julius wrapper class with Julius-Lib libraries.

You can test the SampleApp by the following procedure. At the main window, open the jconf file you want to run Julius with from the menu. After loading the jconf file, execute a start command from the menu. Julius engine will start inside the application as a child thread, and will send messages to the main window at each speech event (trigger, recognition result, etc.).

If you have some trouble displaying the results, try modifying the locale setting at line 98 of SampleApp.cpp to match your language model and re-compile.

The log output of Julius enging will be stored to "`juliuslog.txt`". Please check it if you encounter engine error.

2.4.4.4 How to configure on MSVC

MSVC-based compilation does not use the "**configure**" scheme. If you want to change the configuration options, you should set/unset them manually in the header files below, located at `msvc/config`:

- `config-msvc-julius.h`
- `config-msvc-libjulius.h`
- `config-msvc-libsent.h`

They are copy of "`config.h`" files which will be generated by the configure scripts:

- `julius/config.h`
- `libjulius/config.h`
- `libsent/libsent.h`

To change the configuration, you can first execute the **configure** command on other platform like linux or cygwin, and then look at the generated files above to modify the corresponding MSVC header files.

Chapter 3

Audio Input

Julius accepts waveform input and extracted feature vector input. Waveform data can be given as either a audio file recording speech, or live audio stream via a capture device. You can also use a feature vector input in HTK format.

This chapter describes the specification of audio input in Julius and related tools. For more details about runtime options relating audio input, see the "Audio input" section of the reference manual.

3.1 Audio Format

3.1.1 Number of bits

Quantization bits of the input speech should be 16 bit. No support for 8bit or 24bit input currently.

3.1.2 Number of channels

Number of channels in the recorded data should be one. On live recognition via microphone, the device should support 1 channel recording. Exception is that if you are using OSS interface on Linux (`-input oss`) and only 2-channel recording (stereo) is available, Julius tries to record with the two channel and use only its left channel data.

3.1.3 Sampling Rates

The sampling rate of the input should be given explicitly. The default sampling rate if no option was given is 16,000 Hz. Option `-smpFreq` or `-smpPeriod` can be used to specify the sampling rate in either Hz or 100ns unit respectively. Another way is to use `-htkconf` option to give Julius the HTK Config file you used at AM training, in which case the value of `SOURCERATE` in the Config file will be set.

You should give the correct sampling rate based on the acoustic model you are going to use for recognition. The sampling rate of the input should be the same as the training condition of the acoustic model.

If you are going to use multiple acoustic models with different acoustic conditions, their sampling rate should be the same. You should give the same sampling rate parameters for all the acoustic models and (if you have) GMMs. For more details, see the next chapter about feature extraction.

The given sampling rate will work as requirement to the input. If you use a kind of live input like microphone capture, the given sampling rate will be set to the device and capturing will begin with the sampling rate. Julius will gets error when the sampling rate is not supported on the device. On the other hand, if you are recognizing an audio file, the sampling frequency of the input file is examined against the given sampling rate, and will be rejected if they do not match. [1]

3.2 File input

Option `-input rawfile` tells Julius to read an audio input from file. You can give a file name to be processed to the standard input of Julius. Multiple files can be processed one by one by listing the file names to a text file and specify it by `-filelist`

[1] Please note that this sampling rate check does not work at RAW file input, since RAW file has no header information.

By default, Julius will assume one file as one sentence utterance, with silence part at the beginning and end of the file. But you can apply voice activity detection, silence cutting and other functions normally used for the microphone input by specifying some options. You can also use a Julius function called "short-pause segmentation" to do successive recognition of a long audio stream. See the corresponding chapter of this book for details.

3.2.1 Supported format

Julius can read the following audio file format by default:

- Microsoft WAVE format WAV file (16bit, PCM (no compression), monaural)
- RAW file: no header, signed short (16bit), Big Endian, monoral

If you use `libsndfile` with Julius, you can use additional formats like AU, NIST, ADPCM and so on. The `libsndfile` will be use in Julius if you have libsndfile development files (headers and libraries) in your system when you compile a Julius from source.

You may pay some attentions to the RAW file format. Julius accepts only Big Endian format. If you give Little Endian format RAW file, Julius cannot detect it and outputs wrong result with no warning. You can convert the endianness using sox like this:

```
% sox -t .raw -s -w -c 1 infile -t .raw -s -w -c 1 -x outfile
```

Also you should be careful whether the RAW file has correct data (sampling rate etc.) for the acoustic model you use, since RAW file does not have any header information in itself and Julius can not check them automatically.

3.3 Live microphone input

Option `-input mic` will tell Julius to get the audio input from a raw audio device like microphone or line input. This feature is OS dependent, and supported in Linux, Windows, Mac OS X, FreeBSD and Solaris. [2]

Detection of spoken region from continous input will be performed prior to the main recognition task. By default, a sound input will be detected by a simple level-based detection (level and zero-cross threshold), and then real-time recognition will be performed for each detected region. See the chapter of voice activity detection to tune the detection, or using advanced feature like GMM-based detection. Also see the notes for the real-time recognition.

3.3.1 Preparing microphone input

Julius does not handle any mixer setting of the machine. You should properly set its mixer setting such as recording volumes or capture device (microphone / line) etc.

The recording quality GREATLY affects the recognition performance. Less distortions and less noises will improve the accuracy. Also you should set a proper volume to avoid clipping at a loud voice.

You can check how Julius listens the input audio. If you have a runnning Julius, the best way is to specify an option `-record dir` to save the processed audio data per sentence into files. Another way is to use the tools in Julius distribution, adinrec and adintool, to record audio. They use the same function with Julius, so what they recorded is what Julius will hear.

3.3.2 Notes for supported OS / devices

3.3.2.1 Linux

Julius has two sound API interface for Linux:

- ALSA
- OSS

When specifying `-input mic`, Julius uses ALSA interface to capture audio. You can still explicitly specify which API to use by using option `-input ALSA` or `-input oss`.

[2] You can extend Julius to add support for any input way by an A/D-in plugin. See the chapter of plugin to know how to develop it.

The sound card should support 16-bit recording. Julius uses monaural (1-channel) recording by default, but if you are using OSS interface and only have stereo recording, Julius will recognize its left channel. You can also use USB audio devices.

Another devices can be selected by defining environmental variables. When using ALSA interface (this is default), the default device name string is "default". The device name can be altered by environment variable ALSADEV, for example, if you have multiple audio device and set ALSADEV="plughw:1,0", Julius will listen to the second sound card. When using OSS interface the default device name is /dev/dsp, and it can be changed by the environmental variable AUDIODEV.

3.3.2.2 Windows

On Windows, Julius uses DirectSound API via PortAudio library.

When using Portaudio V19, device will be searched in order of ASIO， DirectSound and MME. The record device can be specified by the environmental variable PORTAUDIO_DEV. When using portaudio v19, the instruction will be output into the log at audio initialization.

3.3.2.3 Mac OS

On Mac OS X, Julius uses CoreAudio API. It is confirmed to run on Mac OS X v10.3.9 and v10.4.1.

3.3.2.4 FreeBSD

On FreeBSD, Julius used the standard snd driver. If compilation fails, try --with-mictype=oss.

3.3.2.5 Sun Solaris

On Sun Solaris, the default device name is /dev/audio. It can be changed by setting an environment variable AUDIODEV. Unlike other OS, Julius on Solaris will automatically change the recording device to microphone. (It is an old feature of early development of Julius)

3.3.3 About Input Delay

You may encounter a time delay on audio input and may want to minimize it. This section describes the reason and show some method to improve it.

Since Most OS are not real-time system, Audio input is oftern buffered per a small chunk (or fragment) at kernel side. When a chunk is filled by the capture device, it will be transmitted to the user process. So the input will delay for the length of the chunk. You can set the size of a chunk by the environment variable LATENCY_MSEC (the value should be milliseconds, not the byte size!). The default value is dependent on OS, and will be output to the tty at startup time. Setting smaller value will decrease the delay, but CPU load will gets higher and may slow down the whole system.

3.4 Network and Socket inputs

3.4.1 original

-input adinnet makes Julius to receive audio stream via network socket. The protocol is a specific one sending just a sequence of audio sample streams per a small packet. There are no detailed document for the procotol, but it's a basic and very simple one, since it has no encription or encode/decode features.

adintool implements the protocol. You can test the adintool like this:

```
Run Julius with network input (it will stop for waiting connection)
% julius .... -input adinnet -freq srate
Run adintool to send audio to the Julius
(server_hostname should be the host where
the abobe Julius is running)
% adintool -in mic -out adinnet -server server_hostname -freq srate
```

3.4.2 esd

-input esd tells Julius to get audio input via EsounD daemon (esd) is supported on Linux. esd is an audio daemon used to share audio I/O among multiple applications. For more details, see the esd manual.

3.4.3 standard

Option `-input stdin` makes Julius to read input from standard input. Only RAW file format is supported with this option.

3.4.4 DATLINK/NetAudio

Julius supports reading direct input from DATLINK server. To use this feature, compile Julius with DATLINK/NetAudio libraries and headers and specify `-input netaudio`. See the Installation chapter to see how to specify the libraries.

3.5 Feature vector file input

Julius can read a feature vector file already extracted from a speech data by other applications such as HTK. You can use this feature to recognize with acoustic features unsupported by Julius. Supported file format is HTK feature file format.

`-input htkparam` or `-input mfcfile` tells Julius to read the file as feature vector file. Like audio file input, multiple file names can be given by listing file names one at a line into a text and specify the file by `-filelist`.

Given an input as feature vector file, its feature type is examined against the acoustic model you are going to use. When they does not match, Julius first checks the difference. If their base forms (basic types) are the same and only the qualifier below is different, Julius modifies the input vector to match the acoustic model and use it, else Julius outputs error and ignore the input.

- addition / removal of delta coef. (_D)
- addition / removal of accelleration coef. (_A)
- supression of energy (_N)

Please note that this checking can be disabled by the option `-notypecheck`

3.6 Audio I/O Extension by Plugin

Julius ver.4.1 and later is capable of extending its audio interface by external plugin. When your target OS is not supported by Julius, or you want add some network-based input into Julius, you can develop a plugin to enable it. See the chapter describing plugin development for more details.

Appendix A

Major Changes

This is a brief summary of big changes between major revisions of recent Julius. The details of each release changes are listed in the file `Release.txt` which is included in the distribution package.

A.1 Changes from 4.0 to 4.1

- Support for Plug-in extension
- Support for multi-stream AM
- Support for MSD-HMM
- Support for CVN and VTLN (`-cvn`, `-vtln`)
- Added output compatibility option (`-fallback1pass`)
- On Linux, default audio API is moved from OSS to ALSA.
- On Linux, audio API can be changed at run time: `-input alsa`, `oss`, `esd`
- Fixed bug: `-multigramout`, environment variable expansion in jconf file, `-record` and others.
- Add option `-usepower` to use power instead of magnitude on MFCC computation.
- This document.

A.2 Changes from 3.5.3 to 4.0

Compatibility issues:

- Julian was merged to Julius. No change for usage, just swap julian to julius.
- Word graph output is now a run-time option(`-lattice`)
- Short-pause segmentation is now a run-time option (`-spsegment`). Also, pause model list can be specified by `-pausemodels` option.
- Multi-path mode integrated, Julius will automatically switch to multipath mode when the AM requires it.
- Module mode extended: new outputs like `<STARTRECOG>`, `<ENDRECOG>`, and new command like `GRAMINFO` and many commands manipulating each recognition process on multi-model recognition.
- Dictionary now allows to omit the output string on the second column. When omitted, Julius uses the LM entry string (firts column) as output string. This is the same format as HTK.
- Dictaionary allows to use double-quotes to quote LM string.

New features:

- Multi-model recognition (`-AM`, `-LM`, `-SR`, `-AM_GMM`, `-inactive`)

- Output each recognition result to a separate file (`-outfile`)
- Log to file instead of stdout, or stop log (`-logfile` / `-nolog`)
- Allow environment variable in jconf file ("`$VARNAME`")
- Down sampling from 48kHz to 16kHz (`-48`)
- Environment variable to set delay time in adin device `LATENCY_MSEC`
- Environment variable to specify capture device name in ALSA: `ALSADEV`
- Rejection based on average power (`-powerthres`, `--enable-power-reject`)
- GMM-based VAD (`--enable-gmm-vad`, `-gmmmargin`, `-gmmup`, `-gmmdown`)
- Decoder-based VAD (`--enable-decoder-vad` `-spdelay`)
- Can specify list of silence models for short-pause segmentation decision (`-pausemodels`)
- Support N-gram longer than 4-gram
- Support recognition with forward only or backward only N-gram.
- Initial support for user-defined LM
- Support isolated word recognition using only dictionary (`-w`, `-wlist`, `-wsil`)
- Confusion network output (`-confnet`)

A.3 Changes from 3.5 to 3.5.3

- Speed up by 20% to 40%, greatly reduced memory access, many fixes on Windows.
- Grammar tools added: dfa_minimize, dfa_determinize, and another tool slf2dfa on Web
- Extended support for MFCC extraction: full parameter settings, MAP-CMN and online energy coef.
- Can read MFCC parameter setting from HTK Config file, and can embed the parameters into binary HMM file.

A.4 Changes from 3.4.2 to 3.5

- Input rejection based on GMM.
- Word lattice output.
- Multi-grammar recognition: `-multigramout`, `-gram`, `-gramlist`
- Character set conversion on output: `-charconv`
- Change input audio device via environmental variable "`AUDIODEV`"
- Now use integrated zlib library to expand gzipped files.
- Integrate all variants of Julius (Linux / Windows / Multi-path ...) into one source tree, and support for compilation with MinGW.
- Almost full documentation of source codes for Doxygen.

Appendix B

Options

All parameters of Julius, including models, paramters and various configurations, should be configured by "options". Options can be specified as command line arguments, or you can write the options into a text file and specify it with "`-C`" argument. The text file that contains Julius options is called "jconf configuration file".

On applications using JuliusLib as recognition engine, the core engine parameters should be set by the options. You can configure the engine in JuliusLib by preparing a jconf configuration file describing all needed options, and pass it to the function `j_config_load_file_new(char *jconffile)`.

When specifying file paths in a jconf configuration file, please be aware that relative paths in jconf file are treated as relative to the jconf file itself, not the current working directory.

Below is the list of all options and its explanations. They are grouped by its belonging class: application options, global options, instance declaration options, LM options, AM and feature options and search options.

B.1 Julius application option

These are application options of Julius, outside of JuliusLib. It contains parameters and switches for result output, character set conversion, log level, and module mode options. These option are specific to Julius, and cannot be used at applications using JuliusLib other than Julius.

`-outfile` On file input, this option write the recognition result of each file to a separate file. The output file of an input file will be the same name but the suffix will be changed to ".out". (rev.4.0)

`-separatescore` Output the language and acoustic scores separately.

`-callbackdebug` Print the callback names at each call for debug. (rev.4.0)

`-charconv` ***`from to`*** Print with character set conversion. *`from`* is the source character set used in the language model, and *`to`* is the target character set you want to get.

On Linux, the arguments should be a code name. You can obtain the list of available code names by invoking the command "iconv --list". On Windows, the arguments should be a code name or codepage number. Code name should be one of "ansi", "mac", "oem", "utf-7", "utf-8", "sjis", "euc". Or you can specify any codepage number supported at your environment.

`-nocharconv` Disable character conversion.

`-module [port]` Run Julius on "Server Module Mode". After startup, Julius waits for tcp/ip connection from client. Once connection is established, Julius start communication with the client to process incoming commands from the client, or to output recognition results, input trigger information and other system status to the client. The default port number is 10500.

`-record` ***`dir`*** Auto-save all input speech data into the specified directory. Each segmented inputs are recorded each by one. The file name of the recorded data is generated from system time when the input ends, in a style of `YYYY.MMDD.HHMMSS.wav`. File format is 16bit monoral WAV. Invalid for mfcfile input.

With input rejection by `-rejectshort`, the rejected input will also be recorded even if they are rejected.

`-logfile` ***`file`*** Save all log output to a file instead of standard output. (Rev.4.0)

-nolog Disable all log output. (Rev.4.0)

-help Output help message and exit.

B.2 Global options

These are model-/search-dependent options relating audio input, sound detection, GMM, decoding algorithm, plugin facility, and others. Global options should be placed before any instance declaration (`-AM`, `-LM`, or `-SR`), or just after "`-GLOBAL`" option.

B.2.1 Audio input

-input {mic|rawfile|mfcfile|adinnet|stdin|netaudio|alsa|oss|esd} Choose speech input source. Specify 'file' or 'rawfile' for waveform file, 'htkparam' or 'mfcfile' for HTK parameter file. On file input, users will be prompted to enter the file name from stdin, or you can use `-filelist` option to specify list of files to process.

'mic' is to get audio input from a default live microphone device, and 'adinnet' means receiving waveform data via tcpip network from an adinnet client. 'netaudio' is from DatLink/NetAudio input, and 'stdin' means data input from standard input.

For waveform file input, only WAV (no compression) and RAW (noheader, 16bit, big endian) are supported by default. Other format can be read when compiled with `libsnd` library. To see what format is actually supported, see the help message using option `-help`. For stdin input, only WAV and RAW is supported. (default: mfcfile)

At Linux, you can choose API at run time by specifying `alsa`, `oss` and `esd`.

-filelist *filename* (With `-input rawfile|mfcfile`) perform recognition on all files listed in the file. The file should contain input file per line. Engine will end when all of the files are processed.

-notypecheck By default, Julius checks the input parameter type whether it matches the AM or not. This option will disable the check and force engine to use the input vector as is.

-48 Record input with 48kHz sampling, and down-sample it to 16kHz on-the-fly. This option is valid for 16kHz model only. The down-sampling routine was ported from sptk. (Rev. 4.0)

-NA *devicename* Host name for DatLink server input (`-input netaudio`).

-adport *port_number* With `-input adinnet`, specify adinnet port number to listen. (default: 5530)

-nostrip Julius by default removes successive zero samples in input speech data. This option inhibits the removal.

-zmean , -nozmean This option enables/disables DC offset removal of input waveform. Offset will be estimated from the whole input. For microphone / network input, zero mean of the first 48000 samples (3 seconds in 16kHz sampling) will be used for the estimation. (default: disabled)

This option uses static offset for the channel. See also `-zmeansource` for frame-wise offset removal.

B.2.2 Speech detection by level and zero-cross

-cutsilence , -nocutsilence Turn on / off the speech detection by level and zero-cross. Default is on for mic / adinnet input, and off for files.

-lv *thres* Level threshold for speech input detection. Values should be in range from 0 to 32767. (default: 2000)

-zc *thres* Zero crossing threshold per second. Only input that goes over the level threshold (`-lv`) will be counted. (default: 60)

-headmargin *msec* Silence margin at the start of speech segment in milliseconds. (default: 300)

-tailmargin *msec* Silence margin at the end of speech segment in milliseconds. (default: 400)

B.2.3 Input rejection

Two simple front-end input rejection methods are implemented, based on input length and average power of detected segment. The rejection by average power is experimental, and can be enabled by `--enable-power-reject` on compilation. Valid for MFCC feature with power coefficient and real-time input only.

For GMM-based input rejection see the GMM section below.

`-rejectshort` ***`msec`*** Reject input shorter than specified milliseconds. Search will be terminated and no result will be output.

`-powerthres` ***`thres`*** Reject the inputted segment by its average energy. If the average energy of the last recognized input is below the threshold, Julius will reject the input. (Rev.4.0)

This option is valid when `--enable-power-reject` is specified at compilation time.

B.2.4 Gaussian mixture model / GMM-VAD

GMM will be used for input rejection by accumulated score, or for front-end GMM-based VAD when `--enable-gmm-vad` is specified.

NOTE: You should also set the proper MFCC parameters required for the GMM, specifying the acoustic parameters described in AM section `-AM_GMM`.

When GMM-based VAD is enabled, the voice activity score will be calculated at each frame as front-end processing. The value will be computed as \[\max_{m \in M_v} p(x|m) - \max_{m \in M_n} p(x|m) \] where M_v is a set of voice GMM, and M_n is a set of noise GMM whose names should be specified by `-gmm-reject`. The activity score will be then averaged for the last N frames, where N is specified by `-gmmmargin`. Julius updates the averaged activity score at each frame, and detect speech up-trigger when the value gets higher than a value specified by `-gmmup`, and detecgt down-trigger when it gets lower than a value of `-gmmdown`.

`-gmm` ***`hmmdefs_file`*** GMM definition file in HTK format. If specified, GMM-based input verification will be performed concurrently with the 1st pass, and you can reject the input according to the result as specified by `-gmmreject`. The GMM should be defined as one-state HMMs.

`-gmmnum` ***`number`*** Number of Gaussian components to be computed per frame on GMM calculation. Only the N-best Gaussians will be computed for rapid calculation. The default is 10 and specifying smaller value will speed up GMM calculation, but too small value (1 or 2) may cause degradation of identification performance.

`-gmmreject` ***`string`*** Comma-separated list of GMM names to be rejected as invalid input. When recognition, the log likelihoods of GMMs accumulated for the entire input will be computed concurrently with the 1st pass. If the GMM name of the maximum score is within this string, the 2nd pass will not be executed and the input will be rejected.

`-gmmmargin` ***`frames`*** (GMM_VAD) Head margin in frames. When a speech trigger detected by GMM, recognition will start from current frame minus this value. (Rev.4.0)

This option will be valid only if compiled with `--enable-gmm-vad`.

`-gmmup` ***`value`*** (GMM_VAD) Up trigger threshold of voice activity score. (Rev.4.1)

This option will be valid only if compiled with `--enable-gmm-vad`.

`-gmmdown` ***`value`*** (GMM_VAD) Down trigger threshold of voice activity score. (Rev.4.1)

This option will be valid only if compiled with `--enable-gmm-vad`.

B.2.5 Decoding switches

Real-time processing means concurrent processing of MFCC computation 1st pass decoding. By default, real-time processing on the pass is on for microphone / adinnet / netaudio input, and for others.

`-realtime , -norealtime` Explicitly switch on / off real-time (pipe-line) processing on the first pass. The default is off for file input, and on for microphone, adinnet and NetAudio input. This option relates to the way CMN and energy normalization is performed: if off, they will be done using average features of whole input. If on, MAP-CMN and energy normalization to do real-time processing.

B.2.6 Misc. options

-C ***jconffile*** Load a jconf file at here. The content of the jconffile will be expanded at this point.

-version Print version information to standard error, and exit.

-setting Print engine setting information to standard error, and exit.

-quiet Output less log. For result, only the best word sequence will be printed.

-debug (For debug) output enormous internal message and debug information to log.

-check {wchmm|trellis|triphone} For debug, enter interactive check mode.

-plugindir ***dirlist*** Specify directory to load plugin. If several direcotries exist, specify them by colon-separated list.

B.3 Instance declaration for multi decoding

The following arguments will create a new configuration set with default parameters, and switch current set to it. Jconf parameters specified after the option will be set into the current set.

To do multi-model decoding, these argument should be specified at the first of each model / search instances with different names. Any options before the first instance definition will be IGNORED.

When no instance definition is found (as older version of Julius), all the options are assigned to a default instance named `_default`.

Please note that decoding with a single LM and multiple AMs is not fully supported. For example, you may want to construct the jconf file as following.

```
-AM am_1 -AM am_2
-LM lm (LM spec..)
-SR search1 am_1 lm
-SR search2 am_2 lm
```

This type of model sharing is not supported yet, since some part of LM processing depends on the assigned AM. Instead, you can get the same result by defining the same LMs for each AM, like this:

```
-AM am_1 -AM am_2
-LM lm_1 (LM spec..)
-LM lm_2 (same LM spec..)
-SR search1 am_1 lm_1
-SR search2 am_2 lm_2
```

-AM ***name*** Create a new AM configuration set, and switch current to the new one. You should give a unique name. (Rev.4.0)

-LM ***name*** Create a new LM configuration set, and switch current to the new one. You should give a unique name. (Rev.4.0)

-SR ***name am_name lm_name*** Create a new search configuration set, and switch current to the new one. The specified AM and LM will be assigned to it. The *am_name* and *lm_name* can be either name or ID number. You should give a unique name. (Rev.4.0)

-AM_GMM When using GMM for front-end processing, you can specify GMM-specific acoustic parameters after this option. If you does not specify `-AM_GMM` with GMM, the GMM will share the same parameter vector as the last AM. The current AM will be switched to the GMM one, so be careful not to confuse with normal AM configurations. (Rev.4.0)

-GLOBAL Start a global section. The global options should be placed before any instance declaration, or after this option on multiple model recognition. This can be used multiple times. (Rev.4.1)

-nosectioncheck , -sectioncheck Disable / enable option location check in multi-model decoding. When enabled, the options between instance declaration is treated as "sections" and only the belonging option types can be written. For example, when an option `-AM` is specified, only the AM related option can be placed after the option until other declaration is found. Also, global options should be placed at top, before any instance declarataion. This is enabled by default. (Rev.4.1)

B.4 Language model (-LM)

This group contains options for model definition of each language model type. When using multiple LM, one instance can have only one LM.

Only one type of LM can be specified for a LM configuration. If you want to use multi model, you should define them one as a new LM.

B.4.1 N-gram

-d *bingram_file* Use binary format N-gram. An ARPA N-gram file can be converted to Julius binary format by mkbingram.

-nlr *arpa_ngram_file* A forward, left-to-right N-gram language model in standard ARPA format. When both a forward N-gram and backward N-gram are specified, Julius uses this forward 2-gram for the 1st pass, and the backward N-gram for the 2nd pass.

Since ARPA file often gets huge and requires a lot of time to load, it may be better to convert the ARPA file to Julius binary format by mkbingram. Note that if both forward and backward N-gram is used for recognition, they together will be converted to a single binary.

When only a forward N-gram is specified by this option and no backward N-gram specified by `-nrl`, Julius performs recognition with only the forward N-gram. The 1st pass will use the 2-gram entry in the given N-gram, and The 2nd pass will use the given N-gram, with converting forward probabilities to backward probabilities by Bayes rule. (Rev.4.0)

-nrl *arpa_ngram_file* A backward, right-to-left N-gram language model in standard ARPA format. When both a forward N-gram and backward N-gram are specified, Julius uses the forward 2-gram for the 1st pass, and this backward N-gram for the 2nd pass.

Since ARPA file often gets huge and requires a lot of time to load, it may be better to convert the ARPA file to Julius binary format by mkbingram. Note that if both forward and backward N-gram is used for recognition, they together will be converted to a single binary.

When only a backward N-gram is specified by this option and no forward N-gram specified by `-nlr`, Julius performs recognition with only the backward N-gram. The 1st pass will use the forward 2-gram probability computed from the backward 2-gram using Bayes rule. The 2nd pass fully use the given backward N-gram. (Rev.4.0)

-v *dict_file* Word dictionary file.

-silhead *word_string* -siltail *word_string* Silence word defined in the dictionary, for silences at the beginning of sentence and end of sentence. (default: "<s>", "</s>")

-mapunk *word_string* Specify unknown word. Default is "<unk>" or "<UNK>". This will be used to assign word probability on unknown words, i.e. words in dictionary that are not in N-gram vocabulary.

-iwspword Add a word entry to the dictionary that should correspond to inter-word pauses. This may improve recognition accuracy in some language model that has no explicit inter-word pause modeling. The word entry to be added can be changed by `-iwspentry`.

-iwspentry *word_entry_string* Specify the word entry that will be added by `-iwspword`. (default: "<UNK> [sp] sp sp")

-sepnum *number* Number of high frequency words to be isolated from the lexicon tree, to ease approximation error that may be caused by the one-best approximation on 1st pass. (default: 150)

B.4.2 Grammar

Multiple grammars can be specified by repeating `-gram` and `-gramlist`. Note that this is unusual behavior from other options (in normal Julius option, last one will override previous ones). You can use `-nogram` to reset the grammars already specified before the point.

-gram gramprefix1[,gramprefix2[,gramprefix3,...]] Comma-separated list of grammars to be used. the argument should be a prefix of a grammar, i.e. if you have `foo.dfa` and `foo.dict`, you should specify them with a single argument `foo`. Multiple grammars can be specified at a time as a comma-separated list.

-gramlist *list_file* Specify a grammar list file that contains list of grammars to be used. The list file should contain the prefixes of grammars, each per line. A relative path in the list file will be treated as relative to the file, not the current path or configuration file.

-dfa *dfa_file* -v *dict_file* An old way of specifying grammar files separately. This is bogus, and should not be used any more.

-nogram Remove the current list of grammars already specified by `-gram`, `-gramlist`, `-dfa` and `-v`.

B.4.3 Isolated word

Dictionary can be specified by using `-w` and `-wlist`. When you specify multiple times, all of them will be read at startup. You can use `-nogram` to reset the already specified dictionaries at that point.

-w *dict_file* Word dictionary for isolated word recognition. File format is the same as other LM. (Rev.4.0)

-wlist *list_file* Specify a dictionary list file that contains list of dictionaries to be used. The list file should contain the file name of dictionaries, each per line. A relative path in the list file will be treated as relative to the list file, not the current path or configuration file. (Rev.4.0)

-nogram Remove the current list of dictionaries already specified by `-w` and `-wlist`.

-wsil *head_sil_model_name tail_sil_model_name sil_context_name* On isolated word recognition, silence models will be appended to the head and tail of each word at recognition. This option specifies the silence models to be appended. *sil_context_name* is the name of the head sil model and tail sil model as a context of word head phone and tail phone. For example, if you specify `-wsil silB silE sp`, a word with phone sequence `b eh t` will be translated as `silB sp-b+eh b-eh+t eh-t+sp silE`. (Rev.4.0)

B.4.4 User-defined LM

-userlm Declare to use user LM functions in the program. This option should be specified if you use user-defined LM functions. (Rev.4.0)

B.4.5 Misc. LM options

-forcedict Skip error words in dictionary and force running.

B.5 Acoustic model and feature analysis (-AM) (-AM_GMM)

This section is about options for acoustic model, feature extraction, feature normalizations and spectral subtraction.

After `-AM name`, an acoustic model and related specification should be written. You can use multiple AMs trained with different MFCC types. For GMM, the required parameter condition should be specified just as same as AMs after `-AM_GMM`.

When using multiple AMs, the values of `-smpPeriod`, `-smpFreq`, `-fsize` and `-fshift` should be the same among all AMs.

B.5.1 Acoustic HMM

-h *hmmdef_file* Acoustic HMM definition file. It should be in HTK ascii format, or Julius binary format. You can convert HTK ascii format to Julius binary format using mkbinhmm.

-hlist *hmmlist_file* HMMList file for phone mapping. This file provides mapping between logical triphone names generated in the dictionary and the defined HMM names in hmmdefs. This option should be specified for context-dependent model.

-tmix ***number*** Specify the number of top Gaussians to be calculated in a mixture codebook. Small number will speed up the acoustic computation, but AM accuracy may get worse with too small value. See also -gprune. (default: 2)

-spmodel ***name*** Specify HMM model name that corresponds to short-pause in an utterance. The short-pause model name will be used in recognition: short-pause skipping on grammar recognition, word-end short-pause model insertion with -iwsp on N-gram, or short-pause segmentation (-spsegment). (default: "sp")

-multipath Enable multi-path mode. To make decoding faster, Julius by default impose a limit on HMM transitions that each model should have only one transition from initial state and to end state. On multi-path mode, Julius does extra handling on inter-model transition to allows model-skipping transition and multiple output/input transitions. Note that specifying this option will make Julius a bit slower, and the larger beam width may be required.

This function was a compilation-time option on Julius 3.x, and now becomes a run-time option. By default (without this option), Julius checks the transition type of specified HMMs, and enable the multi-path mode if required. You can force multi-path mode with this option. (rev.4.0)

-gprune {safe|heuristic|beam|none|default} Set Gaussian pruning algorithm to use. For tied-mixture model, Julius performs Gaussian pruning to reduce acoustic computation, by calculating only the top N Gaussians in each codebook at each frame. The default setting will be set according to the model type and engine setting. default will force accepting the default setting. Set this to none to disable pruning and perform full computation. safe guarantees the top N Gaussians to be computed. heuristic and beam do more aggressive computational cost reduction, but may result in small loss of accuracy model (default: safe (standard), beam (fast) for tied mixture model, none for non tied-mixture model).

-iwcd1 {max|avg|best number} Select method to approximate inter-word triphone on the head and tail of a word in the first pass.

max will apply the maximum likelihood of the same context triphones. avg will apply the average likelihood of the same context triphones. best number will apply the average of top N-best likelihoods of the same context triphone.

Default is best 3 for use with N-gram, and avg for grammar and word. When this AM is shared by LMs of both type, latter one will be chosen.

-iwsppenalty ***float*** Insertion penalty for word-end short pauses appended by -iwsp.

-gshmm ***hmmdef_file*** If this option is specified, Julius performs Gaussian Mixture Selection for efficient decoding. The hmmdefs should be a monophone model generated from an ordinary monophone HMM model, using mkgshmm.

-gsnum ***number*** On GMS, specify number of monophone states to compute corresponding triphones in detail. (default: 24)

B.5.2 Speech analysis

Only MFCC feature extraction is supported in current Julius. Thus when recognizing a waveform input from file or microphone, AM must be trained by MFCC. The parameter condition should also be set as exactly the same as the training condition by the options below.

When you give an input in HTK Parameter file, you can use any parameter type for AM. In this case Julius does not care about the type of input feature and AM, just read them as vector sequence and match them to the given AM. Julius only checks whether the parameter types are the same. If it does not work well, you can disable this checking by -notypecheck.

In Julius, the parameter kind and qualifiers (as TARGETKIND in HTK) and the number of cepstral parameters (NUMCEPS) will be set automatically from the content of the AM header, so you need not specify them by options.

Other parameters should be set exactly the same as training condition. You can also give a HTK Config file which you used to train AM to Julius by -htkconf. When this option is applied, Julius will parse the Config file and set appropriate parameter.

You can further embed those analysis parameter settings to a binary HMM file using mkbinhmm.

If options specified in several ways, they will be evaluated in the order below. The AM embedded parameter will be loaded first if any. Then, the HTK config file given by -htkconf will be parsed. If a value already set

by AM embedded value, HTK config will override them. At last, the direct options will be loaded, which will override settings loaded before. Note that, when the same options are specified several times, later will override previous, except that -htkconf will be evaluated first as described above.

-smpPeriod *period* Sampling period of input speech, in unit of 100 nanoseconds. Sampling rate can also be specified by -smpFreq. Please note that the input frequency should be set equal to the training conditions of AM. (default: 625, corresponds to 16,000Hz)

This option corresponds to the HTK Option SOURCERATE. The same value can be given to this option.

When using multiple AM, this value should be the same among all AMs.

-smpFreq *Hz* Set sampling frequency of input speech in Hz. Sampling rate can also be specified using --smpPeriod. Please note that this frequency should be set equal to the training conditions of AM. (default: 16,000)

When using multiple AM, this value should be the same among all AMs.

-fsize *sample_num* Window size in number of samples. (default: 400)

This option corresponds to the HTK Option WINDOWSIZE, but value should be in samples (HTK value / smpPeriod).

When using multiple AM, this value should be the same among all AMs.

-fshift *sample_num* Frame shift in number of samples. (default: 160)

This option corresponds to the HTK Option TARGETRATE, but value should be in samples (HTK value / smpPeriod).

When using multiple AM, this value should be the same among all AMs.

-preemph *float* Pre-emphasis coefficient. (default: 0.97)

This option corresponds to the HTK Option PREEMCOEF. The same value can be given to this option.

-fbank *num* Number of filterbank channels. (default: 24)

This option corresponds to the HTK Option NUMCHANS. The same value can be given to this option. Be aware that the default value not the same as in HTK (22).

-ceplif *num* Cepstral liftering coefficient. (default: 22)

This option corresponds to the HTK Option CEPLIFTER. The same value can be given to this option.

-rawe , -norawe Enable/disable using raw energy before pre-emphasis (default: disabled)

This option corresponds to the HTK Option RAWENERGY. Be aware that the default value differs from HTK (enabled at HTK, disabled at Julius).

-enormal , -noenormal Enable/disable normalizing log energy. On live input, this normalization will be approximated from the average of last input. (default: disabled)

This option corresponds to the HTK Option ENORMALISE. Be aware that the default value differs from HTK (enabled at HTK, disabled at Julius).

-escale *float_scale* Scaling factor of log energy when normalizing log energy. (default: 1.0)

This option corresponds to the HTK Option ESCALE. Be aware that the default value differs from HTK (0.1).

-silfloor *float* Energy silence floor in dB when normalizing log energy. (default: 50.0)

This option corresponds to the HTK Option SILFLOOR.

-delwin *frame* Delta window size in number of frames. (default: 2)

This option corresponds to the HTK Option DELTAWINDOW. The same value can be given to this option.

-accwin *frame* Acceleration window size in number of frames. (default: 2)

This option corresponds to the HTK Option ACCWINDOW. The same value can be given to this option.

-hifreq ***Hz*** Enable band-limiting for MFCC filterbank computation: set upper frequency cut-off. Value of -1 will disable it. (default: -1)

This option corresponds to the HTK Option HIFREQ. The same value can be given to this option.

-lofreq ***Hz*** Enable band-limiting for MFCC filterbank computation: set lower frequency cut-off. Value of -1 will disable it. (default: -1)

This option corresponds to the HTK Option LOFREQ. The same value can be given to this option.

-zmeanframe , -nozmeanframe With speech input, this option enables/disables frame-wise DC offset removal. This corresponds to HTK configuration ZMEANSOURCE. This cannot be used together with -zmean. (default: disabled)

-usepower Use power instead of magnitude on filterbank analysis. (default: disabled)

B.5.3 Normalization

Julius can perform cepstral mean normalization (CMN) for inputs. CMN will be activated when the given AM was trained with CMN (i.e. has "_Z" qualifier in the header).

The cepstral mean will be estimated in different way according to the input type. On file input, the mean will be computed from the whole input. On live input such as microphone and network input, the ceptral mean of the input is unknown at the start. So MAP-CMN will be used. On MAP-CMN, an initial mean vector will be applied at the beginning, and the mean vector will be smeared to the mean of the incrementing input vector as input goes. Options below can control the behavior of MAP-CMN.

-cvn Enable cepstral variance normalization. At file input, the variance of whole input will be calculated and then applied. At live microphone input, variance of the last input will be applied. CVN is only supported for an audio input.

-vtln ***alpha lowcut hicut*** Do frequency warping, typically for a vocal tract length normalization (VTLN). Arguments are warping factor, high frequency cut-off and low freq. cut-off. They correspond to HTK Config values, WARPFREQ, WARPHCUTOFF and WARPLCUTOFF.

-cmnload ***file*** Load initial cepstral mean vector from file on startup. The *file* should be one saved by -cmnsave. Loading an initial cepstral mean enables Julius to better recognize the first utterance on a real-time input. When used together with -cmnnoupdate, this initial value will be used for all input.

-cmnsave ***file*** Save the calculated cepstral mean vector into *file*. The parameters will be saved at each input end. If the output file already exists, it will be overridden.

-cmnupdate -cmnnoupdate Control whether to update the cepstral mean at each input on real-time input. Disabling this and specifying -cmnload will make engine to always use the loaded static initial cepstral mean.

-cmnmapweight ***float*** Specify the weight of initial cepstral mean for MAP-CMN. Specify larger value to retain the initial cepstral mean for a longer period, and smaller value to make the cepstral mean rely more on the current input. (default: 100.0)

B.5.4 Front-end processing

Julius can perform spectral subtraction to reduce some stationary noise from audio input. Though it is not a powerful method, but it may work on some situation. Julius has two ways to estimate noise spectrum. One way is to assume that the first short segment of an speech input is noise segment, and estimate the noise spectrum as the average of the segment. Another way is to calculate average spectrum from noise-only input using other tool mkss, and load it in Julius. The former one is popular for speech file input, and latter should be used in live input. The options below will switch / control the behavior.

-sscalc Perform spectral subtraction using head part of each file as silence part. The head part length should be specified by -sscalclen. Valid only for file input. Conflict with -ssload.

-sscalclen ***msec*** With -sscalc, specify the length of head silence for noise spectrum estimation in milliseconds. (default: 300)

-ssload ***file*** Perform spectral subtraction for speech input using pre-estimated noise spectrum loaded from *file*. The noise spectrum file can be made by mkss. Valid for all speech input. Conflict with -sscalc.

-ssalpha ***float*** Alpha coefficient of spectral subtraction for -sscalc and -ssload. Noise will be subtracted stronger as this value gets larger, but distortion of the resulting signal also becomes remarkable. (default: 2.0)

-ssfloor ***float*** Flooring coefficient of spectral subtraction. The spectral power that goes below zero after subtraction will be substituted by the source signal with this coefficient multiplied. (default: 0.5)

B.5.5 Misc. AM options

-htkconf ***file*** Parse the given HTK Config file, and set corresponding parameters to Julius. When using this option, the default parameter values are switched from Julius defaults to HTK defaults.

B.6 Recognition process and search (-SR)

This section contains options for search parameters on the 1st / 2nd pass such as beam width and LM weights, configurations for short-pause segmentation, switches for word lattice output and confusion network output, forced alignments, and other options relating recognition process and result output.

Default values for beam width and LM weights will change according to compile-time setup of JuliusLib , AM model type, and LM size. Please see the startup log for the actual values.

B.6.1 1st pass parameters

-lmp ***weight penalty*** (N-gram) Language model weights and word insertion penalties for the first pass.

-penalty1 ***penalty*** (Grammar) word insertion penalty for the first pass. (default: 0.0)

-b ***width*** Beam width in number of HMM nodes for rank beaming on the first pass. This value defines search width on the 1st pass, and has dominant effect on the total processing time. Smaller width will speed up the decoding, but too small value will result in a substantial increase of recognition errors due to search failure. Larger value will make the search stable and will lead to failure-free search, but processing time will grow in proportion to the width.

The default value is dependent on acoustic model type: 400 (monophone), 800 (triphone), or 1000 (triphone, setup=v2.1)

-nlimit ***num*** Upper limit of token per node. This option is valid when --enable-wpair and --enable-wpair-nlimit are enabled at compilation time.

-progout Enable progressive output of the partial results on the first pass.

-proginterval ***msec*** Set the time interval for -progout in milliseconds. (default: 300)

B.6.2 2nd pass parameters

-lmp2 ***weight penalty*** (N-gram) Language model weights and word insertion penalties for the second pass.

-penalty2 ***penalty*** (Grammar) word insertion penalty for the second pass. (default: 0.0)

-b2 ***width*** Envelope beam width (number of hypothesis) at the second pass. If the count of word expansion at a certain hypothesis length reaches this limit while search, shorter hypotheses are not expanded further. This prevents search to fall in breadth-first-like situation stacking on the same position, and improve search failure mostly for large vocabulary condition. (default: 30)

-sb ***float*** Score envelope width for enveloped scoring. When calculating hypothesis score for each generated hypothesis, its trellis expansion and Viterbi operation will be pruned in the middle of the speech if score on a frame goes under the width. Giving small value makes the second pass faster, but computation error may occur. (default: 80.0)

-s ***num*** Stack size, i.e. the maximum number of hypothesis that can be stored on the stack during the search. A larger value may give more stable results, but increases the amount of memory required. (default: 500)

-m ***count*** Number of expanded hypotheses required to discontinue the search. If the number of expanded hypotheses is greater then this threshold then, the search is discontinued at that point. The larger this value is, The longer Julius gets to give up search. (default: 2000)

-n ***num*** The number of candidates Julius tries to find. The search continues till this number of sentence hypotheses have been found. The obtained sentence hypotheses are sorted by score, and final result is displayed in the order (see also the `-output`). The possibility that the optimum hypothesis is correctly found increases as this value gets increased, but the processing time also becomes longer. The default value depends on the engine setup on compilation time: 10 (standard) or 1 (fast or v2.1)

-output ***num*** The top N sentence hypothesis to be output at the end of search. Use with `-n` (default: 1)

-lookuprange ***frame*** Set the number of frames before and after to look up next word hypotheses in the word trellis on the second pass. This prevents the omission of short words, but with a large value, the number of expanded hypotheses increases and system becomes slow. (default: 5)

-looktrellis (Grammar) Expand only the words survived on the first pass instead of expanding all the words predicted by grammar. This option makes second pass decoding faster especially for large vocabulary condition, but may increase deletion error of short words. (default: disabled)

B.6.3 Short-pause segmentation / decoder-VAD

When compiled with `--enable-decoder-vad`, the short-pause segmentation will be extended to support decoder-based VAD.

-spsegment Enable short-pause segmentation mode. Input will be segmented when a short pause word (word with only silence model in pronunciation) gets the highest likelihood at certain successive frames on the first pass. When detected segment end, Julius stop the 1st pass at the point, perform 2nd pass, and continue with next segment. The word context will be considered among segments. (Rev.4.0)

When compiled with `--enable-decoder-vad`, this option enables decoder-based VAD, to skip long silence.

-spdur ***frame*** Short pause duration length to detect end of input segment, in number of frames. (default: 10)

-pausemodels ***string*** A comma-separated list of pause model names to be used at short-pause segmentation. The word whose pronunciation consists of only the pause models will be treated as "pause word" and used for pause detection. If not specified, name of `-spmodel`, `-silhead` and `-siltail` will be used. (Rev.4.0)

-spmargin ***frame*** Back step margin at trigger up for decoder-based VAD. When speech up-trigger found by decoder-VAD, Julius will rewind the input parameter by this value, and start recognition at the point. (Rev.4.0)

This option will be valid only if compiled with `--enable-decoder-vad`.

-spdelay ***frame*** Trigger decision delay frame at trigger up for decoder-based VAD. (Rev.4.0)

This option will be valid only if compiled with `--enable-decoder-vad`.

B.6.4 Word lattice / confusion network output

-lattice , -nolattice Enable / disable generation of word graph. Search algorithm also has changed to optimize for better word graph generation, so the sentence result may not be the same as normal N-best recognition. (Rev.4.0)

-confnet , -noconfnet Enable / disable generation of confusion network. Enabling this will also activates `-lattice` internally. (Rev.4.0)

-graphrange ***frame*** Merge same words at neighbor position at graph generation. If the beginning time and ending time of two word candidates of the same word is within the specified range, they will be merged. The default is 0 (allow merging same words on exactly the same location) and specifying larger value will result in smaller graph output. Setting this value to -1 will disable merging, in that case same words on the same location of different scores will be left as they are. (default: 0)

-graphcut ***depth*** Cut the resulting graph by its word depth at post-processing stage. The depth value is the number of words to be allowed at a frame. Setting to -1 disables this feature. (default: 80)

-graphboundloop ***count*** Limit the number of boundary adjustment loop at post-processing stage. This parameter prevents Julius from blocking by infinite adjustment loop by short word oscillation. (default: 20)

-graphsearchdelay , -nographsearchdelay When this option is enabled, Julius modifies its graph generation algorithm on the 2nd pass not to terminate search by graph merging, until the first sentence candidate is found. This option may improve graph accuracy, especially when you are going to generate a huge word graph by setting broad search. Namely, it may result in better graph accuracy when you set wide beams on both 1st pass -b and 2nd pass -b2, and large number for -n. (default: disabled)

B.6.5 Multi-gram / multi-dic recognition

-multigramout , -nomultigramout On grammar recognition using multiple grammars, Julius will output only the best result among all grammars. Enabling this option will make Julius to output result for each grammar. (default: disabled)

B.6.6 Forced alignment

-walign Do viterbi alignment per word units for the recognition result. The word boundary frames and the average acoustic scores per frame will be calculated.

-palign Do viterbi alignment per phone units for the recognition result. The phone boundary frames and the average acoustic scores per frame will be calculated.

-salign Do viterbi alignment per state for the recognition result. The state boundary frames and the average acoustic scores per frame will be calculated.

B.6.7 Misc. search options

-inactive Start this recognition process instance with inactive state. (Rev.4.0)

-1pass Perform only the first pass.

-fallback1pass When 2nd pass fails, Julius finish the recognition with no result. This option tell Julius to output the 1st pass result as a final result when the 2nd pass fails. Note that some score output (confidence etc.) may not be useful. This was the default behavior of Julius-3.x.

-no_ccd , -force_ccd Explicitly switch phone context handling at search. Normally Julius determines whether the using AM is a context-dependent model or not from the model names, i.e., whether the names contain character + and −. This option will override the automatic detection.

-cmalpha ***float*** Smoothing parameter for confidence scoring. (default: 0.05)

-iwsp (Multi-path mode only) Enable inter-word context-free short pause insertion. This option appends a skippable short pause model for every word end. The short-pause model can be specified by -spmodel.

-transp ***float*** Additional insertion penalty for transparent words. (default: 0.0)

-demo Equivalent to -progout -quiet.

Appendix C

Reference Manuals

C.1 julius

Name

julius – open source multi-purpose LVCSR engine

Synopsis

`julius` [-C *`jconffile`*] [*`options`*...]

Description

julius is a high-performance, multi-purpose, open-source speech recognition engine for researchers and developers. It is capable of performing almost real-time recognition of continuous speech with over 60k-word 3-gram language model and triphone HMM model, on most current PCs. **julius** can perform recognition on audio files, live microphone input, network input and feature parameter files.

The core recognition module is implemented as C library called "JuliusLib". It can also be extended by plug-in facility.

Supported Models

julius needs a language model and an acoustic model to run as a speech recognizer. **julius** supports the following models.

Acoustic model Sub-word HMM (Hidden Markov Model) in HTK ascii format are supported. Phoneme models (monophone), context dependent phoneme models (triphone), tied-mixture and phonetic tied-mixture models of any unit can be used. When using context dependent models, inter-word context dependency is also handled. Multi-stream feature and MSD-HMM is also supported. You can further use a tool **mkbinhmm** to convert the ascii HMM file to a compact binary format for faster loading.

Note that **julius** itself can only extract MFCC features from speech data. If you use acoustic HMM trained for other feature, you should give the input in HTK parameter file of the same feature type.

Language model: word N-gram Word N-gram language model, up to 10-gram, is supported. Julius uses different N-gram for each pass: left-to-right 2-gram on 1st pass, and right-to-left N-gram on 2nd pass. It is recommended to use both LR 2-gram and RL N-gram for Julius. However, you can use only single LR N-gram or RL N-gram. In such case, approximated LR 2-gram computed from the given N-gram will be applied at the first pass.

The Standard ARPA format is supported. In addition, a binary format is also supported for efficiency. The tool mkbingram(1) can convert ARPA format N-gram to binary format.

Language model: grammar The grammar format is an original one, and tools to create a recognirion grammar are included in the distribution. A grammar consists of two files: one is a 'grammar' file that describes sentence structures in a BNF style, using word 'category' name as terminate symbols. Another is a 'voca' file that defines words with its pronunciations (i.e. phoneme sequences) for each category. They should be converted by

mkdfa.pl(1) to a deterministic finite automaton file (.dfa) and a dictionary file (.dict), respectively. You can also use multiple grammars.

Language model: isolated word You can perform isolated word recognition using only word dictionary. With this model type, Julius will perform rapid one pass recognition with static context handling. Silence models will be added at both head and tail of each word. You can also use multiple dictionaries in a process.

Search Algorithm

Recognition algorithm of **julius** is based on a two-pass strategy. Word 2-gram and reverse word 3-gram is used on the respective passes. The entire input is processed on the first pass, and again the final searching process is performed again for the input, using the result of the first pass to narrow the search space. Specifically, the recognition algorithm is based on a tree-trellis heuristic search combined with left-to-right frame-synchronous beam search and right-to-left stack decoding search.

When using context dependent phones (triphones), interword contexts are taken into consideration. For tied-mixture and phonetic tied-mixture models, high-speed acoustic likelihood calculation is possible using gaussian pruning.

For more details, see the related documents.

Options

These options specify the models, system behaviors and various search parameters to Julius. These option can be set at the command line, but it is recommended that you write them in a text file as a "jconf file", and specify it by "-C" option.

Applications incorporating JuliusLib also use these options to set the parameters of core recognition engine. For example, a jconf file can be loaded to the enine by calling `j_config_load_file_new()` with the jconf file name as argument.

Please note that relative paths in a jconf file should be relative to the jconf file itself, not the current working directory.

Below are the details of all options, gathered by group.

Julius application option

These are application options of Julius, outside of JuliusLib. It contains parameters and switches for result output, character set conversion, log level, and module mode options. These option are specific to Julius, and cannot be used at applications using JuliusLib other than Julius.

`-outfile` On file input, this option write the recognition result of each file to a separate file. The output file of an input file will be the same name but the suffix will be changed to ".out". (rev.4.0)

`-separatescore` Output the language and acoustic scores separately.

`-callbackdebug` Print the callback names at each call for debug. (rev.4.0)

`-charconv` ***`from to`*** Print with character set conversion. *`from`* is the source character set used in the language model, and *`to`* is the target character set you want to get.

On Linux, the arguments should be a code name. You can obtain the list of available code names by invoking the command "iconv --list". On Windows, the arguments should be a code name or codepage number. Code name should be one of "ansi", "mac", "oem", "utf-7", "utf-8", "sjis", "euc". Or you can specify any codepage number supported at your environment.

`-nocharconv` Disable character conversion.

`-module [port]` Run Julius on "Server Module Mode". After startup, Julius waits for tcp/ip connection from client. Once connection is established, Julius start communication with the client to process incoming commands from the client, or to output recognition results, input trigger information and other system status to the client. The default port number is 10500.

`-record` ***`dir`*** Auto-save all input speech data into the specified directory. Each segmented inputs are recorded each by one. The file name of the recorded data is generated from system time when the input ends, in a style of `YYYY.MMDD.HHMMSS.wav`. File format is 16bit monoral WAV. Invalid for mfcfile input.

With input rejection by -rejectshort, the rejected input will also be recorded even if they are rejected.

-logfile *file* Save all log output to a file instead of standard output. (Rev.4.0)

-nolog Disable all log output. (Rev.4.0)

-help Output help message and exit.

Global options

These are model-/search-dependent options relating audio input, sound detection, GMM, decoding algorithm, plugin facility, and others. Global options should be placed before any instance declaration (-AM, -LM, or -SR), or just after "-GLOBAL" option.

Audio input

-input {mic|rawfile|mfcfile|adinnet|stdin|netaudio|alsa|oss|esd} Choose speech input source. Specify 'file' or 'rawfile' for waveform file, 'htkparam' or 'mfcfile' for HTK parameter file. On file input, users will be prompted to enter the file name from stdin, or you can use -filelist option to specify list of files to process.

'mic' is to get audio input from a default live microphone device, and 'adinnet' means receiving waveform data via tcpip network from an adinnet client. 'netaudio' is from DatLink/NetAudio input, and 'stdin' means data input from standard input.

For waveform file input, only WAV (no compression) and RAW (noheader, 16bit, big endian) are supported by default. Other format can be read when compiled with libsnd library. To see what format is actually supported, see the help message using option -help. For stdin input, only WAV and RAW is supported. (default: mfcfile)

At Linux, you can choose API at run time by specifying alsa, oss and esd.

-filelist *filename* (With -input rawfile|mfcfile) perform recognition on all files listed in the file. The file should contain input file per line. Engine will end when all of the files are processed.

-notypecheck By default, Julius checks the input parameter type whether it matches the AM or not. This option will disable the check and force engine to use the input vector as is.

-48 Record input with 48kHz sampling, and down-sample it to 16kHz on-the-fly. This option is valid for 16kHz model only. The down-sampling routine was ported from sptk. (Rev. 4.0)

-NA *devicename* Host name for DatLink server input (-input netaudio).

-adport *port_number* With -input adinnet, specify adinnet port number to listen. (default: 5530)

-nostrip Julius by default removes successive zero samples in input speech data. This option inhibits the removal.

-zmean , -nozmean This option enables/disables DC offset removal of input waveform. Offset will be estimated from the whole input. For microphone / network input, zero mean of the first 48000 samples (3 seconds in 16kHz sampling) will be used for the estimation. (default: disabled)

This option uses static offset for the channel. See also -zmeansource for frame-wise offset removal.

Speech detection by level and zero-cross

-cutsilence , -nocutsilence Turn on / off the speech detection by level and zero-cross. Default is on for mic / adinnet input, and off for files.

-lv *thres* Level threshold for speech input detection. Values should be in range from 0 to 32767. (default: 2000)

-zc *thres* Zero crossing threshold per second. Only input that goes over the level threshold (-lv) will be counted. (default: 60)

-headmargin *msec* Silence margin at the start of speech segment in milliseconds. (default: 300)

-tailmargin *msec* Silence margin at the end of speech segment in milliseconds. (default: 400)

Input rejection Two simple front-end input rejection methods are implemented, based on input length and average power of detected segment. The rejection by average power is experimental, and can be enabled by `---enable-power-reject` on compilation. Valid for MFCC feature with power coefficient and real-time input only.

For GMM-based input rejection see the GMM section below.

-rejectshort *msec* Reject input shorter than specified milliseconds. Search will be terminated and no result will be output.

-powerthres *thres* Reject the inputted segment by its average energy. If the average energy of the last recognized input is below the threshold, Julius will reject the input. (Rev.4.0)

This option is valid when `--enable-power-reject` is specified at compilation time.

Gaussian mixture model / GMM-VAD GMM will be used for input rejection by accumulated score, or for front-end GMM-based VAD when `--enable-gmm-vad` is specified.

NOTE: You should also set the proper MFCC parameters required for the GMM, specifying the acoustic parameters described in AM section `-AM_GMM`.

When GMM-based VAD is enabled, the voice activity score will be calculated at each frame as front-end processing. The value will be computed as \[\max_{m \in M_v} p(x|m) - \max_{m \in M_n} p(x|m) \] where M_v is a set of voice GMM, and M_n is a set of noise GMM whose names should be specified by `-gmm-reject`. The activity score will be then averaged for the last N frames, where N is specified by `-gmmmargin`. Julius updates the averaged activity score at each frame, and detect speech up-trigger when the value gets higher than a value specified by `-gmmup`, and detecgt down-trigger when it gets lower than a value of `-gmmdown`.

-gmm *hmmdefs_file* GMM definition file in HTK format. If specified, GMM-based input verification will be performed concurrently with the 1st pass, and you can reject the input according to the result as specified by `-gmmreject`. The GMM should be defined as one-state HMMs.

-gmmnum *number* Number of Gaussian components to be computed per frame on GMM calculation. Only the N-best Gaussians will be computed for rapid calculation. The default is 10 and specifying smaller value will speed up GMM calculation, but too small value (1 or 2) may cause degradation of identification performance.

-gmmreject *string* Comma-separated list of GMM names to be rejected as invalid input. When recognition, the log likelihoods of GMMs accumulated for the entire input will be computed concurrently with the 1st pass. If the GMM name of the maximum score is within this string, the 2nd pass will not be executed and the input will be rejected.

-gmmmargin *frames* (GMM_VAD) Head margin in frames. When a speech trigger detected by GMM, recognition will start from current frame minus this value. (Rev.4.0)

This option will be valid only if compiled with `--enable-gmm-vad`.

-gmmup *value* (GMM_VAD) Up trigger threshold of voice activity score. (Rev.4.1)

This option will be valid only if compiled with `--enable-gmm-vad`.

-gmmdown *value* (GMM_VAD) Down trigger threshold of voice activity score. (Rev.4.1)

This option will be valid only if compiled with `--enable-gmm-vad`.

Decoding option Real-time processing means concurrent processing of MFCC computation 1st pass decoding. By default, real-time processing on the pass is on for microphone / adinnet / netaudio input, and for others.

-realtime , -norealtime Explicitly switch on / off real-time (pipe-line) processing on the first pass. The default is off for file input, and on for microphone, adinnet and NetAudio input. This option relates to the way CMN and energy normalization is performed: if off, they will be done using average features of whole input. If on, MAP-CMN and energy normalization to do real-time processing.

Misc. options

-C ***jconffile*** Load a jconf file at here. The content of the jconffile will be expanded at this point.

-version Print version information to standard error, and exit.

-setting Print engine setting information to standard error, and exit.

-quiet Output less log. For result, only the best word sequence will be printed.

-debug (For debug) output enormous internal message and debug information to log.

-check {wchmm|trellis|triphone} For debug, enter interactive check mode.

-plugindir ***dirlist*** Specify directory to load plugin. If several direcotries exist, specify them by colon-separated list.

Instance declaration for multi decoding

The following arguments will create a new configuration set with default parameters, and switch current set to it. Jconf parameters specified after the option will be set into the current set.

To do multi-model decoding, these argument should be specified at the first of each model / search instances with different names. Any options before the first instance definition will be IGNORED.

When no instance definition is found (as older version of Julius), all the options are assigned to a default instance named `_default`.

Please note that decoding with a single LM and multiple AMs is not fully supported. For example, you may want to construct the jconf file as following.

```
-AM am_1 -AM am_2
-LM lm (LM spec..)
-SR search1 am_1 lm
-SR search2 am_2 lm
```

This type of model sharing is not supported yet, since some part of LM processing depends on the assigned AM. Instead, you can get the same result by defining the same LMs for each AM, like this:

```
-AM am_1 -AM am_2
-LM lm_1 (LM spec..)
-LM lm_2 (same LM spec..)
-SR search1 am_1 lm_1
-SR search2 am_2 lm_2
```

-AM ***name*** Create a new AM configuration set, and switch current to the new one. You should give a unique name. (Rev.4.0)

-LM ***name*** Create a new LM configuration set, and switch current to the new one. You should give a unique name. (Rev.4.0)

-SR ***name am_name lm_name*** Create a new search configuration set, and switch current to the new one. The specified AM and LM will be assigned to it. The *am_name* and *lm_name* can be either name or ID number. You should give a unique name. (Rev.4.0)

-AM_GMM When using GMM for front-end processing, you can specify GMM-specific acoustic parameters after this option. If you does not specify `-AM_GMM` with GMM, the GMM will share the same parameter vector as the last AM. The current AM will be switched to the GMM one, so be careful not to confuse with normal AM configurations. (Rev.4.0)

-GLOBAL Start a global section. The global options should be placed before any instance declaration, or after this option on multiple model recognition. This can be used multiple times. (Rev.4.1)

-nosectioncheck , -sectioncheck Disable / enable option location check in multi-model decoding. When enabled, the options between instance declaration is treated as "sections" and only the belonging option types can be written. For example, when an option `-AM` is specified, only the AM related option can be placed after the option until other declaration is found. Also, global options should be placed at top, before any instance declarataion. This is enabled by default. (Rev.4.1)

Language model (-LM)

This group contains options for model definition of each language model type. When using multiple LM, one instance can have only one LM.

Only one type of LM can be specified for a LM configuration. If you want to use multi model, you should define them one as a new LM.

N-gram

-d *bingram_file* Use binary format N-gram. An ARPA N-gram file can be converted to Julius binary format by mkbingram.

-nlr *arpa_ngram_file* A forward, left-to-right N-gram language model in standard ARPA format. When both a forward N-gram and backward N-gram are specified, Julius uses this forward 2-gram for the 1st pass, and the backward N-gram for the 2nd pass.

Since ARPA file often gets huge and requires a lot of time to load, it may be better to convert the ARPA file to Julius binary format by mkbingram. Note that if both forward and backward N-gram is used for recognition, they together will be converted to a single binary.

When only a forward N-gram is specified by this option and no backward N-gram specified by `-nrl`, Julius performs recognition with only the forward N-gram. The 1st pass will use the 2-gram entry in the given N-gram, and The 2nd pass will use the given N-gram, with converting forward probabilities to backward probabilities by Bayes rule. (Rev.4.0)

-nrl *arpa_ngram_file* A backward, right-to-left N-gram language model in standard ARPA format. When both a forward N-gram and backward N-gram are specified, Julius uses the forward 2-gram for the 1st pass, and this backward N-gram for the 2nd pass.

Since ARPA file often gets huge and requires a lot of time to load, it may be better to convert the ARPA file to Julius binary format by mkbingram. Note that if both forward and backward N-gram is used for recognition, they together will be converted to a single binary.

When only a backward N-gram is specified by this option and no forward N-gram specified by `-nlr`, Julius performs recognition with only the backward N-gram. The 1st pass will use the forward 2-gram probability computed from the backward 2-gram using Bayes rule. The 2nd pass fully use the given backward N-gram. (Rev.4.0)

-v *dict_file* Word dictionary file.

-silhead *word_string* -siltail *word_string* Silence word defined in the dictionary, for silences at the beginning of sentence and end of sentence. (default: "<s>", "</s>")

-mapunk *word_string* Specify unknown word. Default is "<unk>" or "<UNK>". This will be used to assign word probability on unknown words, i.e. words in dictionary that are not in N-gram vocabulary.

-iwspword Add a word entry to the dictionary that should correspond to inter-word pauses. This may improve recognition accuracy in some language model that has no explicit inter-word pause modeling. The word entry to be added can be changed by `-iwspentry`.

-iwspentry *word_entry_string* Specify the word entry that will be added by `-iwspword`. (default: "<UNK> [sp] sp sp")

-sepnum *number* Number of high frequency words to be isolated from the lexicon tree, to ease approximation error that may be caused by the one-best approximation on 1st pass. (default: 150)

Grammar Multiple grammars can be specified by repeating `-gram` and `-gramlist`. Note that this is unusual behavior from other options (in normal Julius option, last one will override previous ones). You can use `-nogram` to reset the grammars already specified before the point.

-gram gramprefix1[,gramprefix2[,gramprefix3,...]] Comma-separated list of grammars to be used. the argument should be a prefix of a grammar, i.e. if you have `foo.dfa` and `foo.dict`, you should specify them with a single argument `foo`. Multiple grammars can be specified at a time as a comma-separated list.

-gramlist ***list_file*** Specify a grammar list file that contains list of grammars to be used. The list file should contain the prefixes of grammars, each per line. A relative path in the list file will be treated as relative to the file, not the current path or configuration file.

-dfa ***dfa_file*** **-v** ***dict_file*** An old way of specifying grammar files separately. This is bogus, and should not be used any more.

-nogram Remove the current list of grammars already specified by `-gram`, `-gramlist`, `-dfa` and `-v`.

Isolated word Dictionary can be specified by using `-w` and `-wlist`. When you specify multiple times, all of them will be read at startup. You can use `-nogram` to reset the already specified dictionaries at that point.

-w ***dict_file*** Word dictionary for isolated word recognition. File format is the same as other LM. (Rev.4.0)

-wlist ***list_file*** Specify a dictionary list file that contains list of dictionaries to be used. The list file should contain the file name of dictionaries, each per line. A relative path in the list file will be treated as relative to the list file, not the current path or configuration file. (Rev.4.0)

-nogram Remove the current list of dictionaries already specified by `-w` and `-wlist`.

-wsil ***head_sil_model_name tail_sil_model_name sil_context_name*** On isolated word recognition, silence models will be appended to the head and tail of each word at recognition. This option specifies the silence models to be appended. *`sil_context_name`* is the name of the head sil model and tail sil model as a context of word head phone and tail phone. For example, if you specify `-wsil silB silE sp`, a word with phone sequence `b eh t` will be translated as `silB sp-b+eh b-eh+t eh-t+sp silE`. (Rev.4.0)

User-defined LM

-userlm Declare to use user LM functions in the program. This option should be specified if you use user-defined LM functions. (Rev.4.0)

Misc. LM options

-forcedict Skip error words in dictionary and force running.

Acoustic model and feature analysis (-AM) (-AM_GMM)

This section is about options for acoustic model, feature extraction, feature normalizations and spectral subtraction.

After `-AM name`, an acoustic model and related specification should be written. You can use multiple AMs trained with different MFCC types. For GMM, the required parameter condition should be specified just as same as AMs after `-AM_GMM`.

When using multiple AMs, the values of `-smpPeriod`, `-smpFreq`, `-fsize` and `-fshift` should be the same among all AMs.

Acoustic HMM

-h ***hmmdef_file*** Acoustic HMM definition file. It should be in HTK ascii format, or Julius binary format. You can convert HTK ascii format to Julius binary format using mkbinhmm.

-hlist ***hmmlist_file*** HMMList file for phone mapping. This file provides mapping between logical triphone names generated in the dictionary and the defined HMM names in hmmdefs. This option should be specified for context-dependent model.

-tmix ***number*** Specify the number of top Gaussians to be calculated in a mixture codebook. Small number will speed up the acoustic computation, but AM accuracy may get worse with too small value. See also `-gprune`. (default: 2)

-spmodel ***name*** Specify HMM model name that corresponds to short-pause in an utterance. The short-pause model name will be used in recognition: short-pause skipping on grammar recognition, word-end short-pause model insertion with `-iwsp` on N-gram, or short-pause segmentation (`-spsegment`). (default: "sp")

-multipath Enable multi-path mode. To make decoding faster, Julius by default impose a limit on HMM transitions that each model should have only one transition from initial state and to end state. On multi-path mode, Julius does extra handling on inter-model transition to allows model-skipping transition and multiple output/input transitions. Note that specifying this option will make Julius a bit slower, and the larger beam width may be required.

This function was a compilation-time option on Julius 3.x, and now becomes a run-time option. By default (without this option), Julius checks the transition type of specified HMMs, and enable the multi-path mode if required. You can force multi-path mode with this option. (rev.4.0)

-gprune {safe|heuristic|beam|none|default} Set Gaussian pruning algorithm to use. For tied-mixture model, Julius performs Gaussian pruning to reduce acoustic computation, by calculating only the top N Gaussians in each codebook at each frame. The default setting will be set according to the model type and engine setting. `default` will force accepting the default setting. Set this to `none` to disable pruning and perform full computation. `safe` guarantees the top N Gaussians to be computed. `heuristic` and `beam` do more aggressive computational cost reduction, but may result in small loss of accuracy model (default: `safe` (standard), `beam` (fast) for tied mixture model, `none` for non tied-mixture model).

-iwcd1 {max|avg|best number} Select method to approximate inter-word triphone on the head and tail of a word in the first pass.

`max` will apply the maximum likelihood of the same context triphones. `avg` will apply the average likelihood of the same context triphones. `best number` will apply the average of top N-best likelihoods of the same context triphone.

Default is `best 3` for use with N-gram, and `avg` for grammar and word. When this AM is shared by LMs of both type, latter one will be chosen.

-iwsppenalty *float* Insertion penalty for word-end short pauses appended by `-iwsp`.

-gshmm *hmmdef_file* If this option is specified, Julius performs Gaussian Mixture Selection for efficient decoding. The hmmdefs should be a monophone model generated from an ordinary monophone HMM model, using mkgshmm.

-gsnum *number* On GMS, specify number of monophone states to compute corresponding triphones in detail. (default: 24)

Speech analysis Only MFCC feature extraction is supported in current Julius. Thus when recognizing a waveform input from file or microphone, AM must be trained by MFCC. The parameter condition should also be set as exactly the same as the training condition by the options below.

When you give an input in HTK Parameter file, you can use any parameter type for AM. In this case Julius does not care about the type of input feature and AM, just read them as vector sequence and match them to the given AM. Julius only checks whether the parameter types are the same. If it does not work well, you can disable this checking by `-notypecheck`.

In Julius, the parameter kind and qualifiers (as `TARGETKIND` in HTK) and the number of cepstral parameters (`NUMCEPS`) will be set automatically from the content of the AM header, so you need not specify them by options.

Other parameters should be set exactly the same as training condition. You can also give a HTK Config file which you used to train AM to Julius by `-htkconf`. When this option is applied, Julius will parse the Config file and set appropriate parameter.

You can further embed those analysis parameter settings to a binary HMM file using mkbinhmm.

If options specified in several ways, they will be evaluated in the order below. The AM embedded parameter will be loaded first if any. Then, the HTK config file given by `-htkconf` will be parsed. If a value already set by AM embedded value, HTK config will override them. At last, the direct options will be loaded, which will override settings loaded before. Note that, when the same options are specified several times, later will override previous, except that `-htkconf` will be evaluated first as described above.

-smpPeriod *period* Sampling period of input speech, in unit of 100 nanoseconds. Sampling rate can also be specified by `-smpFreq`. Please note that the input frequency should be set equal to the training conditions of AM. (default: 625, corresponds to 16,000Hz)

This option corresponds to the HTK Option `SOURCERATE`. The same value can be given to this option.

When using multiple AM, this value should be the same among all AMs.

-smpFreq ***Hz*** Set sampling frequency of input speech in Hz. Sampling rate can also be specified using --smpPeriod. Please note that this frequency should be set equal to the training conditions of AM. (default: 16,000)

When using multiple AM, this value should be the same among all AMs.

-fsize ***sample_num*** Window size in number of samples. (default: 400)

This option corresponds to the HTK Option WINDOWSIZE, but value should be in samples (HTK value / smpPeriod).

When using multiple AM, this value should be the same among all AMs.

-fshift ***sample_num*** Frame shift in number of samples. (default: 160)

This option corresponds to the HTK Option TARGETRATE, but value should be in samples (HTK value / smpPeriod).

When using multiple AM, this value should be the same among all AMs.

-preemph ***float*** Pre-emphasis coefficient. (default: 0.97)

This option corresponds to the HTK Option PREEMCOEF. The same value can be given to this option.

-fbank ***num*** Number of filterbank channels. (default: 24)

This option corresponds to the HTK Option NUMCHANS. The same value can be given to this option. Be aware that the default value not the same as in HTK (22).

-ceplif ***num*** Cepstral liftering coefficient. (default: 22)

This option corresponds to the HTK Option CEPLIFTER. The same value can be given to this option.

-rawe , -norawe Enable/disable using raw energy before pre-emphasis (default: disabled)

This option corresponds to the HTK Option RAWENERGY. Be aware that the default value differs from HTK (enabled at HTK, disabled at Julius).

-enormal , -noenormal Enable/disable normalizing log energy. On live input, this normalization will be approximated from the average of last input. (default: disabled)

This option corresponds to the HTK Option ENORMALISE. Be aware that the default value differs from HTK (enabled at HTK, disabled at Julius).

-escale ***float_scale*** Scaling factor of log energy when normalizing log energy. (default: 1.0)

This option corresponds to the HTK Option ESCALE. Be aware that the default value differs from HTK (0.1).

-silfloor ***float*** Energy silence floor in dB when normalizing log energy. (default: 50.0)

This option corresponds to the HTK Option SILFLOOR.

-delwin ***frame*** Delta window size in number of frames. (default: 2)

This option corresponds to the HTK Option DELTAWINDOW. The same value can be given to this option.

-accwin ***frame*** Acceleration window size in number of frames. (default: 2)

This option corresponds to the HTK Option ACCWINDOW. The same value can be given to this option.

-hifreq ***Hz*** Enable band-limiting for MFCC filterbank computation: set upper frequency cut-off. Value of -1 will disable it. (default: -1)

This option corresponds to the HTK Option HIFREQ. The same value can be given to this option.

-lofreq ***Hz*** Enable band-limiting for MFCC filterbank computation: set lower frequency cut-off. Value of -1 will disable it. (default: -1)

This option corresponds to the HTK Option LOFREQ. The same value can be given to this option.

-zmeanframe , -nozmeanframe With speech input, this option enables/disables frame-wise DC offset removal. This corresponds to HTK configuration ZMEANSOURCE. This cannot be used together with -zmean. (default: disabled)

-usepower Use power instead of magnitude on filterbank analysis. (default: disabled)

Normalization Julius can perform cepstral mean normalization (CMN) for inputs. CMN will be activated when the given AM was trained with CMN (i.e. has "_Z" qualifier in the header).

The cepstral mean will be estimated in different way according to the input type. On file input, the mean will be computed from the whole input. On live input such as microphone and network input, the ceptral mean of the input is unknown at the start. So MAP-CMN will be used. On MAP-CMN, an initial mean vector will be applied at the beginning, and the mean vector will be smeared to the mean of the incrementing input vector as input goes. Options below can control the behavior of MAP-CMN.

-cvn Enable cepstral variance normalization. At file input, the variance of whole input will be calculated and then applied. At live microphone input, variance of the last input will be applied. CVN is only supported for an audio input.

-vtln ***alpha lowcut hicut*** Do frequency warping, typically for a vocal tract length normalization (VTLN). Arguments are warping factor, high frequency cut-off and low freq. cut-off. They correspond to HTK Config values, `WARPFREQ`, `WARPHCUTOFF` and `WARPLCUTOFF`.

-cmnload ***file*** Load initial cepstral mean vector from file on startup. The *file* should be one saved by `-cmnsave`. Loading an initial cepstral mean enables Julius to better recognize the first utterance on a real-time input. When used together with `-cmnnoupdate`, this initial value will be used for all input.

-cmnsave ***file*** Save the calculated cepstral mean vector into *file*. The parameters will be saved at each input end. If the output file already exists, it will be overridden.

-cmnupdate -cmnnoupdate Control whether to update the cepstral mean at each input on real-time input. Disabling this and specifying `-cmnload` will make engine to always use the loaded static initial cepstral mean.

-cmnmapweight ***float*** Specify the weight of initial cepstral mean for MAP-CMN. Specify larger value to retain the initial cepstral mean for a longer period, and smaller value to make the cepstral mean rely more on the current input. (default: 100.0)

Front-end processing Julius can perform spectral subtraction to reduce some stationary noise from audio input. Though it is not a powerful method, but it may work on some situation. Julius has two ways to estimate noise spectrum. One way is to assume that the first short segment of an speech input is noise segment, and estimate the noise spectrum as the average of the segment. Another way is to calculate average spectrum from noise-only input using other tool mkss, and load it in Julius. The former one is popular for speech file input, and latter should be used in live input. The options below will switch / control the behavior.

-sscalc Perform spectral subtraction using head part of each file as silence part. The head part length should be specified by `-sscalclen`. Valid only for file input. Conflict with `-ssload`.

-sscalclen ***msec*** With `-sscalc`, specify the length of head silence for noise spectrum estimation in milliseconds. (default: 300)

-ssload ***file*** Perform spectral subtraction for speech input using pre-estimated noise spectrum loaded from *file*. The noise spectrum file can be made by mkss. Valid for all speech input. Conflict with `-sscalc`.

-ssalpha ***float*** Alpha coefficient of spectral subtraction for `-sscalc` and `-ssload`. Noise will be subtracted stronger as this value gets larger, but distortion of the resulting signal also becomes remarkable. (default: 2.0)

-ssfloor ***float*** Flooring coefficient of spectral subtraction. The spectral power that goes below zero after subtraction will be substituted by the source signal with this coefficient multiplied. (default: 0.5)

Misc. AM options

-htkconf ***file*** Parse the given HTK Config file, and set corresponding parameters to Julius. When using this option, the default parameter values are switched from Julius defaults to HTK defaults.

Recognition process and search (-SR)

This section contains options for search parameters on the 1st / 2nd pass such as beam width and LM weights, configurations for short-pause segmentation, switches for word lattice output and confusion network output, forced alignments, and other options relating recognition process and result output.

Default values for beam width and LM weights will change according to compile-time setup of JuliusLib , AM model type, and LM size. Please see the startup log for the actual values.

1st pass parameters

-lmp ***weight penalty*** (N-gram) Language model weights and word insertion penalties for the first pass.

-penalty1 ***penalty*** (Grammar) word insertion penalty for the first pass. (default: 0.0)

-b ***width*** Beam width in number of HMM nodes for rank beaming on the first pass. This value defines search width on the 1st pass, and has dominant effect on the total processing time. Smaller width will speed up the decoding, but too small value will result in a substantial increase of recognition errors due to search failure. Larger value will make the search stable and will lead to failure-free search, but processing time will grow in proportion to the width.

The default value is dependent on acoustic model type: 400 (monophone), 800 (triphone), or 1000 (triphone, setup=v2.1)

-nlimit ***num*** Upper limit of token per node. This option is valid when `--enable-wpair` and `--enable-wpair-nlimit` are enabled at compilation time.

-progout Enable progressive output of the partial results on the first pass.

-proginterval ***msec*** Set the time interval for `-progout` in milliseconds. (default: 300)

2nd pass parameters

-lmp2 ***weight penalty*** (N-gram) Language model weights and word insertion penalties for the second pass.

-penalty2 ***penalty*** (Grammar) word insertion penalty for the second pass. (default: 0.0)

-b2 ***width*** Envelope beam width (number of hypothesis) at the second pass. If the count of word expansion at a certain hypothesis length reaches this limit while search, shorter hypotheses are not expanded further. This prevents search to fall in breadth-first-like situation stacking on the same position, and improve search failure mostly for large vocabulary condition. (default: 30)

-sb ***float*** Score envelope width for enveloped scoring. When calculating hypothesis score for each generated hypothesis, its trellis expansion and Viterbi operation will be pruned in the middle of the speech if score on a frame goes under the width. Giving small value makes the second pass faster, but computation error may occur. (default: 80.0)

-s ***num*** Stack size, i.e. the maximum number of hypothesis that can be stored on the stack during the search. A larger value may give more stable results, but increases the amount of memory required. (default: 500)

-m ***count*** Number of expanded hypotheses required to discontinue the search. If the number of expanded hypotheses is greater then this threshold then, the search is discontinued at that point. The larger this value is, The longer Julius gets to give up search. (default: 2000)

-n ***num*** The number of candidates Julius tries to find. The search continues till this number of sentence hypotheses have been found. The obtained sentence hypotheses are sorted by score, and final result is displayed in the order (see also the `-output`). The possibility that the optimum hypothesis is correctly found increases as this value gets increased, but the processing time also becomes longer. The default value depends on the engine setup on compilation time: 10 (standard) or 1 (fast or v2.1)

-output ***num*** The top N sentence hypothesis to be output at the end of search. Use with `-n` (default: 1)

-lookuprange ***frame*** Set the number of frames before and after to look up next word hypotheses in the word trellis on the second pass. This prevents the omission of short words, but with a large value, the number of expanded hypotheses increases and system becomes slow. (default: 5)

-looktrellis (Grammar) Expand only the words survived on the first pass instead of expanding all the words predicted by grammar. This option makes second pass decoding faster especially for large vocabulary condition, but may increase deletion error of short words. (default: disabled)

Short-pause segmentation / decoder-VAD When compiled with `--enable-decoder-vad`, the short-pause segmentation will be extended to support decoder-based VAD.

-spsegment Enable short-pause segmentation mode. Input will be segmented when a short pause word (word with only silence model in pronunciation) gets the highest likelihood at certain successive frames on the first pass. When detected segment end, Julius stop the 1st pass at the point, perform 2nd pass, and continue with next segment. The word context will be considered among segments. (Rev.4.0)

When compiled with `--enable-decoder-vad`, this option enables decoder-based VAD, to skip long silence.

-spdur *frame* Short pause duration length to detect end of input segment, in number of frames. (default: 10)

-pausemodels *string* A comma-separated list of pause model names to be used at short-pause segmentation. The word whose pronunciation consists of only the pause models will be treated as "pause word" and used for pause detection. If not specified, name of `-spmodel`, `-silhead` and `-siltail` will be used. (Rev.4.0)

-spmargin *frame* Back step margin at trigger up for decoder-based VAD. When speech up-trigger found by decoder-VAD, Julius will rewind the input parameter by this value, and start recognition at the point. (Rev.4.0)

This option will be valid only if compiled with `--enable-decoder-vad`.

-spdelay *frame* Trigger decision delay frame at trigger up for decoder-based VAD. (Rev.4.0)

This option will be valid only if compiled with `--enable-decoder-vad`.

Word lattice / confusion network output

-lattice , -nolattice Enable / disable generation of word graph. Search algorithm also has changed to optimize for better word graph generation, so the sentence result may not be the same as normal N-best recognition. (Rev.4.0)

-confnet , -noconfnet Enable / disable generation of confusion network. Enabling this will also activates `-lattice` internally. (Rev.4.0)

-graphrange *frame* Merge same words at neighbor position at graph generation. If the beginning time and ending time of two word candidates of the same word is within the specified range, they will be merged. The default is 0 (allow merging same words on exactly the same location) and specifying larger value will result in smaller graph output. Setting this value to `-1` will disable merging, in that case same words on the same location of different scores will be left as they are. (default: 0)

-graphcut *depth* Cut the resulting graph by its word depth at post-processing stage. The depth value is the number of words to be allowed at a frame. Setting to -1 disables this feature. (default: 80)

-graphboundloop *count* Limit the number of boundary adjustment loop at post-processing stage. This parameter prevents Julius from blocking by infinite adjustment loop by short word oscillation. (default: 20)

-graphsearchdelay , -nographsearchdelay When this option is enabled, Julius modifies its graph generation algorithm on the 2nd pass not to terminate search by graph merging, until the first sentence candidate is found. This option may improve graph accuracy, especially when you are going to generate a huge word graph by setting broad search. Namely, it may result in better graph accuracy when you set wide beams on both 1st pass `-b` and 2nd pass `-b2`, and large number for `-n`. (default: disabled)

Multi-gram / multi-dic recognition

-multigramout , -nomultigramout On grammar recognition using multiple grammars, Julius will output only the best result among all grammars. Enabling this option will make Julius to output result for each grammar. (default: disabled)

Forced alignment

-walign Do viterbi alignment per word units for the recognition result. The word boundary frames and the average acoustic scores per frame will be calculated.

-palign Do viterbi alignment per phone units for the recognition result. The phone boundary frames and the average acoustic scores per frame will be calculated.

-salign Do viterbi alignment per state for the recognition result. The state boundary frames and the average acoustic scores per frame will be calculated.

Misc. search options

-inactive Start this recognition process instance with inactive state. (Rev.4.0)

-1pass Perform only the first pass.

-fallback1pass When 2nd pass fails, Julius finish the recognition with no result. This option tell Julius to output the 1st pass result as a final result when the 2nd pass fails. Note that some score output (confidence etc.) may not be useful. This was the default behavior of Julius-3.x.

-no_ccd , -force_ccd Explicitly switch phone context handling at search. Normally Julius determines whether the using AM is a context-dependent model or not from the model names, i.e., whether the names contain character + and −. This option will override the automatic detection.

-cmalpha ***float*** Smoothing parameter for confidence scoring. (default: 0.05)

-iwsp (Multi-path mode only) Enable inter-word context-free short pause insertion. This option appends a skippable short pause model for every word end. The short-pause model can be specified by `-spmodel`.

-transp ***float*** Additional insertion penalty for transparent words. (default: 0.0)

-demo Equivalent to `-progout -quiet`.

ENVIRONMENT VARIABLES

ALSADEV (using mic input with alsa device) specify a capture device name. If not specified, "`default`" will be used.

AUDIODEV (using mic input with oss device) specify a capture device path. If not specified, "`/dev/dsp`" will be used.

PORTAUDIO_DEV (portaudio V19) specify the name of capture device to use. See the instruction output of log at start up how to specify it.

LATENCY_MSEC Try to set input latency of microphone input in milliseconds. Smaller value will shorten latency but sometimes make process unstable. Default value will depend on the running OS.

EXAMPLES

For examples of system usage, refer to the tutorial section in the Julius documents.

NOTICE

Note about jconf files: relative paths in a jconf file are interpreted as relative to the jconf file itself, not to the current directory.

SEE ALSO

julian(1), jcontrol(1), adinrec(1), adintool(1), mkbingram(1), mkbinhmm(1), mkgsmm(1), wav2mfcc(1), mkss(1)
http://julius.sourceforge.jp/en/

DIAGNOSTICS

Julius normally will return the exit status 0. If an error occurs, Julius exits abnormally with exit status 1. If an input file cannot be found or cannot be loaded for some reason then Julius will skip processing for that file.

BUGS

There are some restrictions to the type and size of the models Julius can use. For a detailed explanation refer to the Julius documentation. For bug-reports, inquires and comments please contact julius-info at lists.sourceforge.jp.

COPYRIGHT

AUTHORS

Rev.1.0 (1998/02/20) Designed by Tatsuya KAWAHARA and Akinobu LEE (Kyoto University)

Development by Akinobu LEE (Kyoto University)

Rev.1.1 (1998/04/14), Rev.1.2 (1998/10/31), Rev.2.0 (1999/02/20), Rev.2.1 (1999/04/20), Rev.2.2 (1999/10/04), Rev.3.0 (2000/02/14),
Development of above versions by Akinobu LEE (Kyoto University)

Rev.3.2 (2001/08/15), Rev.3.3 (2002/09/11), Rev.3.4 (2003/10/01), Rev.3.4.1 (2004/02/25), Rev.3.4.2 (2004/04/30)
Development of above versions by Akinobu LEE (Nara Institute of Science and Technology)

Rev.3.5 (2005/11/11), Rev.3.5.1 (2006/03/31), Rev.3.5.2 (2006/07/31), Rev.3.5.3 (2006/12/29), Rev.4.0 (2007/12/19), Rev.4.1 (2008/1
Development of above versions by Akinobu LEE (Nagoya Institute of Technology)

THANKS TO

From rev.3.2, Julius is released by the "Information Processing Society, Continuous Speech Consortium".
The Windows DLL version was developed and released by Hideki BANNO (Nagoya University).
The Windows Microsoft Speech API compatible version was developed by Takashi SUMIYOSHI (Kyoto University).

C.2 jcontrol

Name

jcontrol – a sample module client written in C

Synopsis

`jcontrol` *`hostname`* [*`portnum`*]

Description

jcontrol is a simple console program to control julius running on other host via network API. It can send command to Julius, and receive messages from Julius.

When invoked, jcontrol tries to connect to Julius running in "module mode" on specified hostname. After connection established, jcontrol waits for user commands from standard input.

When user types a command to jcontrol, it will be interpreted and cor- responding API command will be sent to Julius. When a message is received from Julius, its content will be output to standard output.

For the details about the API, see the related documents.

Options

hostname Host name where Julius is runnning in module mode.

portnum port number (default: 10500)

COMMANDS

jcontrol interprets commands from standard input. Below is a list of all commands.

Engine control

pause Stop Julius and enter into paused status. In paused status, Julius will not run recognition even if speech input occurs. When this command is issued while recognition is running, Julius will stop after the recognition has been finished.

terminate Same as `pause`, but discard the current speech input when received command in the middle of recognition process.

resume Restart Julius that has been paused or terminated.

inputparam ***arg*** Tell Julius how to deal with speech input in case grammar is changed just when recognition is running. Specify one: "`TERMINATE`", "`PAUSE`" or "`WAIT`".

version Tell Julius to send version description string.

status Tell Julius to send the system status (`active` / `sleep`)

Grammar handling

graminfo Tell the current process to send the list of current grammars to client.

changegram ***prefix*** Send a new grammar "`prefix.dfa`" and "`prefix.dict`", and tell julius to use it as a new grammar. All the current grammars used in the current process of Julius will be deleted and replaced to the specifed grammar.

On isolated word recognition, the dictionary alone should be specified as "`filename.dict`" instead of *`prefix`*.

addgram ***prefix*** Send a new grammar "`prefix.dfa`" and "`prefix.dict`" and add it to the current grammar.

On isolated word recognition, the dictionary alone should be specified as "`filename.dict`" instead of *`prefix`*.

deletegram ***gramlist*** Tell Julius to delete existing grammar. The grammar can be specified by either prefix name or number ID. The number ID can be determined from the message sent from Julius at each time grammar information has changed. When want to delete more than one grammar, specify all of them as comma-sparated.

deactivategram ***gramlist*** Tell Julius to de-activate a specified grammar. The specified grammar will still be kept but will not be used for recognition.

The target grammar can be specified by either prefix name or number ID. The number ID can be determined from the message sent from Julius at each time grammar information has changed. When want to delete more than one grammar, specify all of them as comma-sparated.

activategram ***gramlist*** Tell Julius to activate previously de-activated grammar. The target grammar can be specified by either prefix name or number ID. The number ID can be determined from the message sent from Julius at each time grammar information has changed. When want to delete more than one grammar, specify all of them as comma-sparated.

addword ***grammar_name_or_id dictfile*** Add the recognition word entries in the specified *`dictfile`* to the specified grammar on current process.

syncgram Force synchronize grammar status, like unix command "sync".

Process management

Julius-4 supports multi-model recognition nad multi decoding. In this case it is possible to control each recognition process, as defined by "-SR" option, from module client.

In multi decoding mode, the module client holds "current process", and the process commands and grammar related commands will be issued toward the current process.

listprocess Tell Julius to send the list of existing recognition process.

currentprocess ***procname*** Switch the current process to the process specified by the name.

shiftprocess Rotate the current process. At each call the current process will be changed to the next one.

addprocess ***jconffile*** Tell Julisu to load a new recognition process into engine. The argument *jconffile* should be a jconf file that contains only one set of LM options and one SR definition. Note that the file should be visible on the running Julius, since **jcontrol** only send the path name and Julius actually read the jconf file.

The new LM and SR process will have the name of the jconffile.

delprocess ***procname*** Delete the specified recognition process from the engine.

deactivateprocess ***procname*** Tell Julius to temporary stop the specified recognition process. The stopped process will not be executed for the input until activated again.

activateprocess ***procname*** Tell Julius to activate the temporarily stopped process.

EXAMPLES

The dump messages from Julius are output to tty with prefix "> " appended to each line. Julius can be started in module mode like this:

```
% julius -C ... -module
```

jcontrolcan be launched with the host name:

```
% jcontrol hostname
```

It will then receive the outputs of Julius and output the raw message to standard out. Also, by inputting the commands above to the standard input of **jcontrol**, it will be sent to Julius. See manuals for the specification of module mode.

SEE ALSO

julius (1)

COPYRIGHT

LICENSE

C.3 jclient.pl

Name

jclient.pl – sample client for module mode (perl version)

Synopsis

`jclient.pl`

Description

This is yet another sample client written in perl. It will connect to Julius running in module mode, receive recognition results from Julius, and cna send commands to control Julius.

This is a tiny program with only 57 lines. You can use it for free.

EXAMPLES

Invoke Julius with module mode by specifying "`-module`" option:

```
% julius -C ... -module
```

Then, at other terminal or other host, invoke **jclient.pl** like below. The default hostname is "`localhost`", and port number is 10500. You can change them by editing the top part of the script.

```
% jclient.pl
```

It will then receive the outputs of Julius and output the raw message to standard out. Also, by inputting a raw module command to the standard input of **jclient.pl**, it will be sent to Julius. See manuals for the specification of module mode.

SEE ALSO

julius (1) , jcontrol (1)

COPYRIGHT

"**jclient.pl**" has been developed by Dr. Ryuichi Nisimura (nisimura@sys.wakayama-u.ac.jp). Use at your own risk.

If you have any feedback, comment or request, please contact the E-mail address above, or look at the Web page below.

http://w3voice.jp/

C.4 mkbingram

Name

mkbingram – make binary N-gram from ARPA N-gram file

Synopsis

`mkbingram` [-nlr *`forward_ngram.arpa`*] [-nrl *`backward_ngram.arpa`*] [-d *`old_bingram_file`*] output_bingram_file

Description

mkbingram is a tool to convert N-gram definition file(s) in ARPA standard format to a compact Julius binary format. It will speed up the initial loading time of N-gram much faster. It can read gzipped file directly.

From rev.4.0, Julius can deal with forward N-gram, backward N-gram and their combinations. So, **mkbingram** now generates binary N-gram file from one of them, or combining them two to produce one binary N-gram.

When only a forward N-gram is specified, **mkbingram** generates binary N-gram from only the forward N-gram. When using this binary N-gram at Julius, it performs the 1st pass with the 2-gram probabilities in the N-gram, and run the 2nd pass with the given N-gram fully, with converting forward probabilities to backward probabilities by Bayes rule.

When only a backward N-gram is specified, **mkbingram** generates an binary N-gram file that contains only the backward N-gram. The 1st pass will use forward 2-gram probabilities that can be computed from the backward 2-gram using Bayes rule, and the 2nd pass use the given backward N-gram fully.

When both forward and backward N-grams are specified, the 2-gram part in the forward N-gram and all backward N-gram will be combined into single bingram file. The forward 2-gram will be applied for the 1st pass and backward N-gram for the 2nd pass. Note that both N-gram should be trained in the same corpus with same parameters (i.e. cut-off thresholds), with same vocabulary.

The old binary N-gram produced by **mkbingram** of version 3.x and earlier can be used in Julius-4, but you can convert the old version to the new version by specifying it as input of current **mkbingram** by option "`-d`".

Please note that binary N-gram file converted by **mkbingram** of version 4.0 and later cannot be read by older Julius 3.x.

Options

`-nlr` ***`forward_ngram.arpa`*** Read in a forward (left-to-right) word N-gram file in ARPA standard format.

`-nrl` ***`backward_ngram.arpa`*** Read in a backward (right-to-left) word N-gram file in ARPA standard format.

`-d` ***`old_bingram_file`*** Read in a binary N-gram file.

`-swap` Swap BOS word <s> and EOS word </s> in N-gram.

`output_bingram_file` binary N-gram file name to output.

EXAMPLES

Convert a set of forward and backward N-gram in ARPA format into Julius binary form:

```
% mkbingram -nlr 2gram.arpa -nrl rev-Ngram.arpa outfile
```

Convert a single forward 4-gram in ARPA format into a binary file:

```
% mkbingram -nlr 4gram.arpa outfile
```

Convert old binary N-gram file to current format:

```
% mkbingram -d old_bingram new_bingram
```

SEE ALSO

julius (1) , mkbinhmm (1) , mkbinhmmlist (1)

COPYRIGHT

LICENSE

C.5 mkbinhmm

Name

mkbinhmm – convert HMM definition file in HTK ascii format to Julius binary format

Synopsis

`mkbinhmm` [-htkconf *`HTKConfigFile`*] hmmdefs_file binhmm_file

Description

mkbinhmm convert an HMM definition file in HTK ascii format into a binary HMM file for Julius. It will greatly speed up the launch process.

You can also embed acoustic analysis condition parameters needed for recognition into the output file. To embed the parameters, specify the HTK Config file you have used to extract acoustic features for training the HMM by the optione "`-htkconf`".

The embedded parameters in a binary HMM format will be loaded into Julius automatically, so you do not need to specify the acoustic feature options at run time. It will be convenient when you deliver an acoustic model.

You can also specify binary file as the input. This can be used to update the old binary format into new one, or to embed the config parameters into the already existing binary files. If the input binhmm already has acoustic analysis parameters embedded, they will be overridden by the specified values.

mkbinhmm can read gzipped file as input.

Options

`-htkconf` ***`HTKConfigFile`*** HTK Config file you used at training time. If specified, the values are embedded to the output file.

`hmmdefs_file` The source HMm definitino file in HTK ascii format or Julius binary format.

`hmmdefs_file` Output file.

EXAMPLES

Convert HTK ascii format HMM definition file into Julius binary file:

```
% mkbinhmm hmmdefs.ascii binhmm
```

Furthermore, embed acoustic feature parameters as specified by Config file

```
% mkbinhmm -htkconf Config hmmdefs.ascii binhmm
```

Embed the acoustic parameters into an existing binary file

```
% mkbingram -htkconf Config old_binhmm new_binhmm
```

SEE ALSO

julius (1) , mkbingram (1) , mkbinhmmlist (1)

COPYRIGHT

LICENSE

C.6 mkbinhmmlist

Name

mkbinhmmlist – convert HMMList file into binary format

Synopsis

`mkbinhmmlist` hmmdefs_file HMMList_file output_binhmmlist_file

Description

mkbinhmmlist converts a HMMList file to binary format. Since the index trees for lookup are also stored in the binary format, it will speed up the startup of Julius, namely when using big HMMList file.

For conversion, HMM definition file *hmmdefs_file* that will be used together at Julius needs to be specified. The format of the HMM definition file can be either ascii or Julius binary format.

The output binary file can be used in Julius as the same by "-hlist". The format wil be auto-detected by Julius.

mkbinhmmlist can read gzipped file.

Options

hmmdefs_file Acoustic HMM definition file, in HMM ascii format or Julius binary format.

HMMList_file Source HMMList file

output_binhmmlist_file Output file, will be overwritten if already exist.

EXAMPLES

Convert a HMMList file `logicalTri` into binary format and store to `logicalTri.bin`:

```
% mkbinhmmlist binhmm logicalTri logicalTri.bin
```

SEE ALSO

julius (1) , mkbinhmm (1)

COPYRIGHT

LICENSE

The same as Julius.

C.7 adinrec

Name

adinrec – record audio device and save one utterance to a file

Synopsis

`adinrec` [*options*...] *filename*

Description

adinrec opens an audio stream, detects an utterance input and store it to a specified file. The utterance detection is done by level and zero-cross thresholds. Default input device is microphone, but other audio input source, including Julius A/D-in plugin, can be used by using "-input" option.

The audio format is 16 bit, 1 channel, in Microsoft WAV format. If the given filename already exists, it will be overridden.

If filename is "-" , the captured data will be streamed into standard out, with no header (raw format).

Options

adinrec uses JuliusLib and adopts Julius options. Below is a list of valid options.

adinrec specific options

-freq ***Hz*** Set sampling rate in Hz. (default: 16,000)

-raw Output in raw file format.

JuliusLib options

-input {mic|rawfile|adinnet|stdin|netaudio|esd|alsa|oss} Choose speech input source. Specify 'file' or 'rawfile' for waveform file. On file input, users will be prompted to enter the file name from stdin.

'mic' is to get audio input from a default live microphone device, and 'adinnet' means receiving waveform data via tcpip network from an adinnet client. 'netaudio' is from DatLink/NetAudio input, and 'stdin' means data input from standard input.

At Linux, you can choose API at run time by specifying `alsa`, `oss` and `esd`.

-lv ***thres*** Level threshold for speech input detection. Values should be in range from 0 to 32767. (default: 2000)

-zc ***thres*** Zero crossing threshold per second. Only input that goes over the level threshold (`-lv`) will be counted. (default: 60)

-headmargin ***msec*** Silence margin at the start of speech segment in milliseconds. (default: 300)

-tailmargin ***msec*** Silence margin at the end of speech segment in milliseconds. (default: 400)

-zmean This option enables DC offset removal.

-smpFreq ***Hz*** Set sampling rate in Hz. (default: 16,000)

-48 Record input with 48kHz sampling, and down-sample it to 16kHz on-the-fly. This option is valid for 16kHz model only. The down-sampling routine was ported from sptk. (Rev. 4.0)

-NA ***devicename*** Host name for DatLink server input (`-input netaudio`).

-adport ***port_number*** With `-input adinnet`, specify adinnet port number to listen. (default: 5530)

-nostrip Julius by default removes successive zero samples in input speech data. This option stop it.

-C ***jconffile*** Load a jconf file at here. The content of the jconffile will be expanded at this point.

-plugindir ***dirlist*** Specify which directories to load plugin. If several direcotries exist, specify them by colon-separated list.

ENVIRONMENT VARIABLES

ALSADEV Device name string for ALSA. (default: "`default`")

AUDIODEV Device name string for OSS. (default: "`/dev/dsp`")

PORTAUDIO_DEV (portaudio V19) specify the name of capture device to use. See the instruction output of log at start up how to specify it.

LATENCY_MSEC Input latency of microphone input in milliseconds. Smaller value will shorten latency but sometimes make process unstable. Default value will depend on the running OS.

SEE ALSO

julius (1) , adintool (1)

COPYRIGHT

LICENSE

C.8 adintool

Name

adintool – a tool to record / split / send / receive audio streams

Synopsis

`adintool` -in *`inputdev`* -out *`outputdev`* [*`options`*...]

Description

adintool analyzes speech input, finds speech segments skipping silence, and records the detected segments in various ways. It performs speech detection based on zerocross number and power (level), and records the detected parts to files or other output devices sucessively.

adintool is a upper version of adinrec with various functions. Supported input device are: microphone input, a speech file, standard tty input, and network socket (called adin-net server mode). Julius plugin can be also used. Detected speech segments will be saved to output devices: speech files, standard tty output, and network socket (called adin-net client mode). For example, you can split the incoming speech to segments and send them to Julius to be recognized.

Output format is WAV, 16bit (signed short), monoral. If the file already exist, it will be overridden.

Options

All Julius options can be set. Only audio input related options are treated and others are silently skipped. Below is a list of options.

adintool specific options

`-freq` ***`Hz`*** Set sampling rate in Hz. (default: 16,000)

`-in` ***`inputdev`*** Audio input device. "mic" to capture via microphone input, "file" for audio file input, and "stdin" to read raw data from standard-input. For file input, file name prompt will appear after startup. Use "adinnet" to make **adintool** as "adinnet server", receiving data from client via network socket. Default port number is 5530, which can be altered by option "`-inport`".

Alternatively, input device can be set by "`-input`" option, in which case you can use plugin input.

`-out` ***`outputdev`*** Audio output device store the data. Specify "file" to save to file, in which the output filename should be given by "`-filename`". Use "stdout" to standard out. "adinnet" will make **adintool** to be an adinnet client, sending speech data to a server via tcp/ip socket. When using "adinnet" output, the server name to send data should be specified by "`-server`". The default port number is 5530, which can be changed by "`-port`" option.

`-inport` ***`num`*** When adintool becomes adinnet server to receive data (-in adinnet), set the port number to listen. (default: 5530)

`-server` **[host] [,host...]** When output to adinnet server (-out adinnet), set the hostname. You can send to multiple hosts by specifying their hostnames as comma-delimited list like "host1,host2,host3".

-port [num] [,num...] When adintool send a data to adinnet server (-out adinnet), set the port number to connect. (default: 5530) For multiple servers, specify port numbers for all servers like "5530,5530,5531".

-filename *file* When output to file (`-out file`), set the output filename. The actual file name will be as "`file.0000.wav`", "`file.0001.wav`" and so on, where the four digit number increases as speech segment detected. The initial number will be set to 0 by default, which can be changed by "`-startid`" option. When using "`-oneshot`" option to save only the first segment, the input will be saved as "`file`".

-startid *number* At file output, set the initial file number. (default: 0)

-oneshot Exit after the end of first speech segment.

-nosegment Do not perform speech detection for input, just treat all the input as a single valid segment.

-raw Output as RAW file (no header).

-autopause When output to adinnet server, **adintool** enter pause state at every end of speech segment. It will restart when the destination adinnet server sends it a resume signal.

-loosesync When output to multiple adinnet server, not to do strict synchronization for restart. By default, when **adintool** has entered pause state, it will not restart until resume commands are received from all servers. This option will allow restart at least one restart command has arrived.

-rewind *msec* When input is a live microphone device, and there has been some continuing input at the moment **adintool** resumes, it start recording backtracking by the specified milliseconds.

Concerning Julius options

-input {mic|rawfile|adinnet|stdin|netaudio|esd|alsa|oss} Choose speech input source. Specify 'file' or 'rawfile' for waveform file. On file input, users will be prompted to enter the file name from stdin.

'mic' is to get audio input from a default live microphone device, and 'adinnet' means receiving waveform data via tcpip network from an adinnet client. 'netaudio' is from DatLink/NetAudio input, and 'stdin' means data input from standard input.

At Linux, you can choose API at run time by specifying `alsa`, `oss` and `esd`.

-lv *thres* Level threshold for speech input detection. Values should be in range from 0 to 32767. (default: 2000)

-zc *thres* Zero crossing threshold per second. Only input that goes over the level threshold (`-lv`) will be counted. (default: 60)

-headmargin *msec* Silence margin at the start of speech segment in milliseconds. (default: 300)

-tailmargin *msec* Silence margin at the end of speech segment in milliseconds. (default: 400)

-zmean This option enables DC offset removal.

-smpFreq *Hz* Set sampling rate in Hz. (default: 16,000)

-48 Record input with 48kHz sampling, and down-sample it to 16kHz on-the-fly. This option is valid for 16kHz model only. The down-sampling routine was ported from sptk. (Rev. 4.0)

-NA *devicename* Host name for DatLink server input (`-input netaudio`).

-adport *port_number* With `-input adinnet`, specify adinnet port number to listen. (default: 5530)

-nostrip Julius by default removes successive zero samples in input speech data. This option stop it.

-C *jconffile* Load a jconf file at here. The content of the jconffile will be expanded at this point.

-plugindir *dirlist* Specify which directories to load plugin. If several direcotries exist, specify them by colon-separated list.

ENVIRONMENT VARIABLES

ALSADEV (using mic input with alsa device) specify a capture device name. If not specified, "default" will be used.

AUDIODEV (using mic input with oss device) specify a capture device path. If not specified, "/dev/dsp" will be used.

PORTAUDIO_DEV (portaudio V19) specify the name of capture device to use. See the instruction output of log at start up how to specify it.

LATENCY_MSEC Try to set input latency of microphone input in milliseconds. Smaller value will shorten latency but sometimes make process unstable. Default value will depend on the running OS.

EXAMPLES

Record microphone input to files: "data.0000.wav", "data.0001.wav" and so on:

```
% adintool -in mic -out file -filename data
```

Split a long speech file "foobar.raw" into "foobar.1500.wav", "foobar.1501.wav" ...:

```
% adintool -in file -out file -filename foobar -startid 1500
% enter filename->foobar.raw
```

Copy an entire audio file via network socket.

```
(sender) % adintool -in file -out adinnet -server receiver_hostname -nosegment
(receiver) % adintool -in adinnet -out file -nosegment
```

Detect speech segment, send to Julius via network and recognize it:

```
(sender) % adintool -in mic -out adinnet -server receiver_hostname
(receiver) % julius -C ... -input adinnet
```

SEE ALSO

julius (1) , adinrec (1)

COPYRIGHT

LICENSE

C.9 mkss

Name

mkss – calculate average spectrum for spectral subtraction

Synopsis

mkss [*options*...] filename

Description

mkss is a tool to estimate noise spectrum for spectral subtraction on Julius. It reads a few seconds of sound data from microphone input, calculate the average spectrum and save it to a file. The output file can be used as a noise spectrum data in Julius (option "`-ssload`").

The recording will start immediately after startup. Sampling format is 16bit, monoral. If outpue file already exist, it will be overridden.

Options

`-freq` ***`Hz`*** Sampling frequency in Hz (default: 16,000)

`-len` ***`msec`*** capture length in milliseconds (default: 3000)

`-fsize` ***`sample_num`*** frame size in number of samples (default: 400)

`-fshift` ***`sample_num`*** frame shift in number of samples (default: 160)

SEE ALSO

julius (1)

COPYRIGHT

LICENSE

C.10 mkgshmm

Name

mkgshmm – convert monophone HMM to GS HMM for Julius

Synopsis

`mkgshmm` monophone_hmmdefs > outputfile

Description

mkgshmm converts monophone HMM definition file in HTK format into a special format for Gaussian Mixture Selection (GMS) in Julius.

GMS is an algorithm to reduce the amount of acoustic computation with triphone HMM, by pre-selection of promising gaussian mixtures using likelihoods of corresponding monophone mixtures.

EXAMPLES

(1) Prepare a monophone model which was trained by the same corpus as target triphone model.
(2) Convert the monophone model using mkgshmm.

```
% mkgshmm monophone > gshmmfile
```

(3) Specify the output file in Julius with option "-gshmm"

```
% julius -C ... -gshmm gshmmfile
```

SEE ALSO

julius (1)

COPYRIGHT

LICENSE

The same as Julius.

C.11 generate-ngram

Name

generate-ngram – random sentence generator from N-gram

Synopsis

`generate-ngram` [*`options`*...] *`binary_ngram`*

Description

generate-ngram is a tool to generate sentences randomly according to the given N-gram language model. The N-gram model file *`binary_ngram`* should be an binary format.

Options

`-n` ***`num`*** Number of sentences to generate (default: 10)

`-N` Specify which length of N-gram to use (default: available max in the given model)

`-bos` Beginning-of-sentence word (default: "<s>")

`-eos` End-of-sentence word (default: "</s>")

`-ignore` Specify a word to be supressed from output (default: "<UNK")

`-v` Verbose output.

`-debug` Debug output.

SEE ALSO

julius (1) , mkbingram (1)

COPYRIGHT

LICENSE

The same as Julius.

C.12 mkdfa.pl

Name

mkdfa.pl – grammar compiler

Synopsis

`mkdfa.pl` [*options*...] prefix

Description

mkdfa.pl compiles the Julian format grammar (`.grammar` and `.voca`) to Julian native formats (`.dfa` and `.dict`). In addition, "`.term`" will be also generated that stores correspondence of category ID used in the output files to the source category name.

`prefix` should be the common file name prefix of "`.grammar`" and "voca" file. From `prefix.grammar` and `prefix.voca` file, `prefix.dfa`, `prefix.dict` and `prefix.term` will be output.

Options

`-n` Not process dictionary. You can only convert `.grammar` file to `.dfa` file without `.voca` file.

ENVIRONMENT VARIABLES

`TMP or TEMP` Set directory to store temporal file. If not specified, one of them on the following list will be used: `/tmp`, `/var/tmp`, `/WINDOWS/Temp`, `/WINNT/Temp`.

EXAMPLES

Convert a grammar `foo.grammar` and `foo.voca` to `foo.dfa`, `foo.voca` and `foo.term`.

```
% mkdfa.pl foo
```

SEE ALSO

julius (1) , generate (1) , nextword (1) , accept_check (1) , dfa_minimize (1)

DIAGNOSTICS

mkdfa.pl invokes **mkfa** and **dfa_minimize** internally. They should be placed at the same directory as **mkdfa.pl**.

COPYRIGHT

LICENSE

C.13 generate

Name

generate – random sentence generator from a grammar

Synopsis

`generate` [-v] [-t] [-n *num*] [-s *spname*] prefix

Description

This small program randomly generates sentences that are acceptable by the given grammar.
`.dfa`, `.dict` and `.term` files are needed to execute. They can be generated from `.grammar` and `.voca` file by **mkdfa.pl**.

Options

`-t` Output in word's category name.

`-n` ***`num`*** Set number of sentences to be generated (default: 10)

`-s` ***`spname`*** the name string of short-pause word to be supressed (default: "`sp`")

`-v` Debug output mode.

EXAMPLES

Exmple output of a sample grammar "`fruit`":

```
% generate fruit
Stat: init_voca: read 36 words
Reading in term file (optional)...done
15 categories, 36 words
DFA has 26 nodes and 42 arcs
-----
 <s> I WANT ONE APPLE </s>
 <s> I WANT TEN PEARS </s>
 <s> CAN I HAVE A PINEAPPLE </s>
 <s> I WANT ONE PEAR </s>
 <s> COULD I HAVE A BANANA </s>
 <s> I WANT ONE APPLE PLEASE </s>
 <s> I WANT NINE APPLES </s>
 <s> NINE APPLES </s>
 <s> I WANT ONE PINEAPPLE </s>
 <s> I WANT A PEAR </s>
```

SEE ALSO

mkdfa.pl (1) , generate-ngram (1)

COPYRIGHT

LICENSE

C.14 nextword

Name

nextword – display next predicted words (in reverse order)

Synopsis

`nextword` [-t] [-r] [-s *spname*] [-v] prefix

Description

Given a partial (part of) sentence from the end, it outputs the next words allowed in the specified grammar.

`.dfa`, `.dict` and `.term` files are needed to execute. They can be generated from `.grammar` and `.voca` file by **mkdfa.pl**.

Please note that the latter part of sentence should be given, since the main 2nd pass does a right-to-left parsing.

Options

- **`-t`** Input / Output in category name. (default: word)
- **`-r`** Enter in reverse order
- **`-s`** ***`spname`*** the name string of short-pause word to be supressed (default: "`sp`")
- **`-v`** Debug output.

EXAMPLES

Exmple output of a sample grammar "`fruit`":

```
% nextword fruit
Stat: init_voca: read 36 words
Reading in term file (optional)...done
15 categories, 36 words
DFA has 26 nodes and 42 arcs
-----
command completion is disabled
-----
wseq > A BANANA </s>
[wseq: A BANANA </s>]
[cate: (NUM_1|NUM_1|A|A) FRUIT_SINGULAR NS_E]
PREDICTED CATEGORIES/WORDS:
                    NS_B (<s> )
                    HAVE (HAVE )
                    WANT (WANT )
                    NS_B (<s> )
                    HAVE (HAVE )
                    WANT (WANT )
```

SEE ALSO

mkdfa.pl (1) , generate (1) , accept_check (1)

COPYRIGHT

LICENSE

C.15 accept_check

Name

accept_check – Check whether a grammar accept / reject given word sequences

Synopsis

`accept_check` [-t] [-s *spname*] [-v] prefix

Description

accept_check is a tool to check whether a sentence can be accepted or rejected on a grammar (`prefix.dfa` and `prefix.dict`). The sentence should be given from standard input. You can do a batch check by preparing all test sentence at each line of a text file, and give it as standard input of **accept_check**.

This tool needs .dfa, .dict and .term files. You should convert a written grammar file to generate them by **mkdfa.pl**.

A sentence should be given as space-separated word sequence. It may be required to add head / tail silence word like `sil`, depending on your grammar. And should not contain a short-pause word.

When a word belongs to various category in a grammar, **accept_check** will check all the possible sentence patterns, and accept it if any of those is acceptable.

Options

-t Use category name as input instead of word.

-s ***spname*** Short-pause word name to be skipped. (default: "`sp`")

-v Debug output.

EXAMPLES

An output for "date" grammar:

```
% echo '<s> NEXT SUNDAY </s>' | accept_check date
Reading in dictionary...
143 words...done
Reading in DFA grammar...done
Mapping dict item <-> DFA terminal (category)...done
Reading in term file (optional)...done
27 categories, 143 words
DFA has 35 nodes and 71 arcs
-----
wseq: <s> NEXT SUNDAY </s>
cate: NS_B (NEXT|NEXT) (DAYOFWEEK|DAYOFWEEK|DAY|DAY) NS_E
accepted
```

SEE ALSO

mkdfa.pl (1) , generate (1) , nextword (1)

COPYRIGHT

LICENSE

C.16 dfa_minimize

Name

dfa_minimize – Minimize a DFA grammar network

Synopsis

`dfa_minimize` [-o *`outfile`*] dfafile

Description

dfa_minimize will convert an .dfa file to an equivalent minimal form. Output to standard output, or to a file specified by "`-o`" option.

On version 3.5.3 and later, **mkdfa.pl** invokes this tool inside, and the output .dfa file will be always minimized, so you do not need to use this manually.

Options

`-o` ***`outfile`*** Output file. If not specified output to standard output.

EXAMPLES

Minimize `foo.dfa` to `bar.dfa`:

```
% dfa_minimize -o bar.dfa foo.dfa
```

Another way:

```
% dfa_minimize < foo.dfa > bar.dfa
```

SEE ALSO

mkdfa.pl (1)

COPYRIGHT

LICENSE

C.17 dfa_determinize

Name

dfa_determinize – Determinize NFA grammar network.

Synopsis

`dfa_determinize` [-o *`outfile`*] dfafile

Description

dfa_determinize converts a non-deterministic .dfa file into deterministic DFA. Output to standard output, or file specified by "-o" option.

This additional tool is not necessary on a grammar building procedure in Julius, since the grammar network generated by **mkdfa.pl** is always determinized.

Options

-o ***outfile*** Outout file. If not specified, output to stdout.

EXAMPLES

Determinize `foo.dfa` to `bar.dfa`:

```
% dfa_determinize -o bar.dfa foo.dfa
```

Another way:

```
% dfa_determinize < foo.dfa > bar.dfa
```

SEE ALSO

mkdfa.pl (1) , dfa_minimize (1)

COPYRIGHT

LICENSE

The same as Julius.

C.18 gram2sapixml.pl

Name

gram2sapixml.pl – convert Julius grammar to SAPI XML grammar format

Synopsis

`gram2sapixml.pl` [*prefix*...]

Description

gram2sapixml.pl converts a recognition grammar file of Julius (.grammar, .voca) to Microsoft SAPI XML grammar format. *prefix* should be a file name of target grammar, excluding suffixes. If multiple argument is given, each will be process sequencialy in turn.

The internal character set should be in UTF-8 format. By default **gram2sapixml.pl** assume input in EUC-JP encoding and tries to convert it to UTF-8 using **iconv**. You may want to disable this feature within the script.

It will fail to convert a left recursive rule in the grammar. When fails, it will leave the source rules in the target .xml file, so you should modify the output manually to solve it.

SEE ALSO

mkdfa.pl (1)

DIAGNOSTICS

The conversion procedure is somewhat dumb one, only converting the non-terminal symbols and terminal symbols (=word category name) into corresponding rules one by one. This is only a help tool, and you will need a manual inspection and editing to use it on a real SAPI application.

COPYRIGHT

LICENSE

Appendix D

License term

Julius is an open-source software provided as is for free. It has its own license term, and its major points can be summarized as:

- No need to make your source code open, and binary distribution is allowed.
- Can be used for commercial use for free, at your own risk.
- When you publish or present any results by using the Software, you must explicitly mention your use of "Large Vocabulary Continuous Speech Recognition Engine Julius".

For more information about the license, please refer to the "`LICENSE.txt`" file included in this archive.