

連続音声認識ソフトウェア Julius

Open-Source Speech Recognition Software Julius

河原達也
Tatsuya Kawahara

京都大学学術情報メディアセンター
Kyoto University, Academic Center and Media Studies.
kawahara@i.kyoto-u.ac.jp <http://www.ar.media.kyoto-u.ac.jp/>

李 晃伸
Akinobu Lee

奈良先端科学技術大学院大学情報科学研究科
Nara Institute of Science and Technology, School of Information Science.
ri@is.naist.jp

Keywords: Speech Recognition

Julius は、ディクテーション（音声入力ワープロのような口述筆記）などの大語彙連続音声認識を主な目的として著者らが開発したフリーソフトウェアである。その後、記述文法への対応を含めて様々な機能拡張・性能改善が継続的に行われており、現在は下記の Web サイトにおいて、最新・完全版が無償で入手できる。

Julius の Web サイト：<http://julius.sourceforge.jp/>

本稿では、特に非専門家を想定してその使用法を解説する。まず 1・2 章で概要を述べた後、3 章及び 4 章で、ディクテーション及びタスク文法を想定した各場合について、一通りの操作法を説明する。次の 5 章で、音声認識と Julius の動作原理について簡単に述べた後、6 章及び 7 章で、Unix 版の主要なオプション、及び Windows SAPI 版の設定について説明する。最後に 8 章で、典型的なトラブルシューティングと今後の課題について述べる。

1. Julius の特徴

（市販の音声認識ソフトと比較して）Julius の最大の特徴は、汎用性・可搬性である。すなわち、音響モデルや言語モデルなどのインタフェースが公開されており、それらを置き換えたり、修正したりするのが容易なことである。これにより、様々な使用環境や目的に応じたシステムの構築が簡単にできる。さらに、ソースコードも公開されているので、システム自体の修正・拡張も可能である。逆に事前登録学習（エンロールメント）や自動適応学習などの機能は現在実装されておらず、単語登録をする GUI もないので、音響モデルや単語辞書・文法ファイルなどを直接操作する必要がある。

第二の特徴は様々なプラットフォームで動作することである。実際に、Linux を含む Unix、Windows、Mac OS などで動作が確認されている。特に Linux で動作する日本語音声認識ソフトは希少である。さらに、Windows XP に搭載されている SAPI (Speech API) に対応した版もある。将来は、PDA やマイコンへの移植も期待される。

Julius は、このように自由度が高いため、パソコン上のディクテーションにとどまらず、講演などの自動書き起こし、会話ロボット・エージェント、家電の操作、次世代カーナビなど実に様々な用途に利用されている。その反面、自由度が高いということは、ある程度中身を理解して、自分で組み立てる必要があることを意味する。例えていうなら、市販のソフトが電器店で販売されている既製のパソコンのようなものであるのに対して、Julius はカスタムメイドのパソコンに対応する。したがって、パーツを理解した上で、自分の目的に合致した仕様のものを選んで構成する必要がある。自分の要求にあう仕様のパーツがなければ、自分で作成することもできるのである。（現実的には専門家（音声認識の研究を行っている大学の研究室など）に依頼した方が確実だろう。）

2. Julius 及び Julian の開発の経緯と派生版

当初 Julius は IPA のプロジェクトである「日本語ディクテーション基本ソフトウェア」のコアエンジンとして作成された。したがって、ディクテーションで一般的な単語 N-gram モデル (N 単語連鎖の統計量に基づくモデル；詳細は 5.2 節参照) のみを扱う仕様であった。参考文献[1]の付録 CD-ROM に収められているものもこの版 (Rev.3.1) である。

一方、音声対話システムなどタスクが明確な場合は専用の文法を人手で記述するのが一般的である。このような記述文法向けの音声認識エンジンとして、(Julius と大半のコードを共有した) Julian が作成された。こちらは IPA プロジェクトの後継である連続音声認識コンソーシアム (<http://www.lang.astem.or.jp/CSRC/>) を通した限定的な配布であったが、2003 年秋にその終了にあわせて、Julian を Julius のパッケージに統合してフリーで公開することとした。このような経緯から、現在でも Unix 版で記述文法用のエンジンの実行形式は "julian" という名前になっている。

Julius にはまたいくつかの派生版が存在する。Unix 版がオリジナルであり、スタンドアローンの音声認識器として動作するほか、認識サーバーとしても動作する。

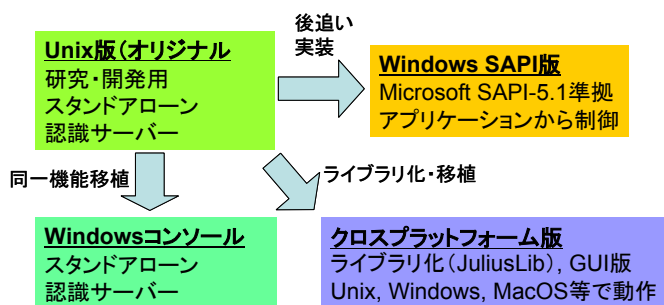


図1 Julius 及び Julian の各種バージョンの関係

Windows SAPI 版は、Microsoft Speech API (SAPI) 用の認識エンジンとして動作し、SAPI-5.1 にほぼ完全に準拠している。そのため、アプリケーションは SAPI 経由で Julius を制御でき、入力デバイスも SAPI が操作するため、アプリケーションシステムの開発が容易である。ただし、Unix 版の後追いで実装しており、すべての機能が利用できるわけではない。Unix 版を (API を用意しないで) そのまま Windows に移植したコンソール版、Julius をライブラリ化したクロスプラットフォーム版 (Windows, Mac OS などで動作) も存在する。これらの関係を図1にまとめる。使用目的に応じて適当なものを選択されるのが望ましいが、音声認識システムの研究開発用には Unix 版を推奨する。以下では、基本的に Unix 版あるいは Windows コンソール版を中心に説明を行い、7章において Windows SAPI 版について解説する。

3. とりあえずディクテーションソフトとして使ってみる

Unix 版の動作環境は、Linux を含む Unix である。Linux 用の実行バイナリがあらかじめ用意されているが、他の Unix についてもソースからコンパイルすることで動作する。(2004年10月時点の最新版は rev.3.4.2)

ディクテーションの実行に必要なもの (Julius 実行バイナリ、N-gram 言語モデル、及び標準的な音響モデル) を1つにまとめた「ディクテーション実行キット」が Web サイト (<http://julius.sourceforge.jp/models.htm>) で公開されている。展開したディレクトリ上で、

```
% ./bin/julius -C fast.jconf
```

を実行することで、マイクからの音声入力に対するディクテーションが行える。Windows コンソール版でも同様である。この動作例を図2に示す。モデルをすべて読み込んで起動した後、入力音声ごとに認識を実行する。“pass1_best” で始まる行が第1パスでの中間結果、“sentence” で始まる行が第2パスを行った後の最終的な認識結果である。単語列のほかにも N-gram エントリ

```
% ./bin/julius -C fast.jconf
include config: fast.jconf
(中略)
----- System Info begin -----
Julius rev.3.4.2 (fast)
(システム設定情報出力、中略)
----- System Info end -----

*****
* NOTICE: The first input may not be correctly recognized *
*           since no CMN parameter is available on startup. *
*****

<<< please speak >>>

pass1_best: 日本は明確なメッセージを出すべきだ。
sentence:   日本は明確なメッセージを出すべきだ。
```

図2 Julius の動作例

列、音素列、仮説スコア、そして単語毎の信頼度を出力することもできる。ここで、“-C” で指定しているのは認識時の設定ファイルである。詳細は6章を参照されたい。

マイク入力の音声認識する際、Julius はマイクデバイスの設定を一切行わないため、マイクから音声が入力できるように、録音デバイスの選択や録音ボリュームの調節を事前に行っておく必要がある。一部のサウンドドライバでは録音品質が著しく低く、認識が困難な場合もある。またマイク入力においては、音響特徴量の正規化 (CMS: ケプストラム平均減算) を前の発話に基づいて行うため、最初の発話に対しては認識精度が低下する可能性がある。

4. タスク文法を書いてみる

機器の操作や天気などの情報案内・商品の注文など、音声認識を用いるタスクが限定されている場合は、想定される発話パターンを文法の形で記述するのが、高い認識精度を実現する上で手っ取り早い。

記述文法の場合は、“julian” という実行形式を用いる。Julian で用いる文法の形式は独自のものであり、文法規則を grammar ファイルに、単語辞書を voca ファイルにそれぞれ記述する。図3に、果物注文タスクにおける想定文を受理する Julian 用の文法と語彙ファイルの例を示す。

これらを付属のコンパイラ “mkdfa.pl” を用いて Julian で使用できる形式 (.dict 及び .dfa) へ変換する。ここで、voca ファイルに単語の読みを直接音素列で記述する必要があるが、ひらがなを音素列へ変換するスクリプト “yomi2voca.pl” が Unix 版パッケージに含まれている。この文法コンパイルの過程も含めた Julian の動作例を図4に示す。設定方法や出力はディクテーションの場合とほぼ同じである。出力に単語のカテゴリ (文法の終端記号) の番号も含めることもできる。このカテゴリ番号は文法コンパイル時に .term ファイルに出力される。

このような文法を記述するのは慣れないと難しいかもしれないが、Julius の Web サイトにいくつかのタスク

grammarファイル(文法規則) "fruit.grammar"

```
S      : NS_B FRUIT_N PLEASE NS_E
FRUIT_N : FRUIT
FRUIT_N : FRUIT NUM KO
FRUIT_N : FRUIT WO NUM KO
PLEASE  : WO KUDASAI
PLEASE  : KUDASAI
PLEASE  : NISHITE KUDASAI
PLEASE  : DESU
```

vocaファイル(単語辞書) "fruit.voca"

```
(左下より)
% FRUIT          % WO
蜜柑      m i k a N   を      o
リンゴ     r i N g o   % KUDASAI
ぶどう     b u d o:   ください k u d a s a i
% NUM          % NISHITE
0          z e r o   にして n i s h i t e
1          i c h i   % DESU
...        % DESU
9          k y u:    です      d e s u
% KO          % NS_B
個         k o      <s>      silB
(右上へ続く)  </s>    silE
```

図3 Julian用文法の例(果物注文タスク)

```
認識用文法のコンパイル
% ./bin/mkdfa.pl fruit
fruit.grammar has 8 rules
fruit.voca has 9 categories and 20 words
---
Now parsing grammar file
Now modifying grammar to minimize states[1]
Now parsing vocabulary file
Now making nondeterministic finite automaton[9/9]
Now making deterministic finite automaton[9/9]
Now making triplet list[9/9]
---
-rw-r--r-- 1 foo bar 152 Sep  2 16:41 fruit.dfa
-rw-r--r-- 1 foo bar 349 Sep  2 16:41 fruit.dict
-rw-r--r-- 1 foo bar  65 Sep  2 16:41 fruit.term

音声ファイルの認識
% ./bin/julian -dfa fruit.dfa -v fruit.dict -C file.jconf
(中略)
enter filename->sample.wav
(中略)
### Recognition: 2nd pass (RL heuristic best-first with DFA)
samplenum=210
sentencel: <s> リンゴ を 3 個 ください </s>
wseq1: 7 0 3 1 2 4 8
phseq1: silB | r i N g o | o | s a N | k o | k u d a s a i | silE
cmscore1: 1.000 1.000 0.750 1.000 0.683 1.000 1.000
score1: -5109.966309
25 generated, 25 pushed, 8 nodes popped in 210
(CTRL-Cで終了)

マイク入力の認識
% ./bin/julian -dfa fruit.dfa -v fruit.dict -C mic.jconf
(中略)
<<< please speak >>>
pass1_best: リンゴ 3 個 ください
sentencel: リンゴ 3 個 を ください
```

図4 Julianの実行例

<http://www.ai-gakkai.or.jp/jsai/journal/toolbox/02/julius.tgz> か
<http://julius.sourceforge.jp/archive/julian-sample-kit.tar.gz> で動作確認可能)

や日時や金額など典型的なサブタスクに対するサンプル文法が用意されているので、参考にされたい。

5. 音声認識と Julius の動作原理

ここまで一通りの使用はできると期待される。ただし、比較的静かな環境でパソコンのマイク入力を用いて、丁寧に発声されることを前提としている。さらに性能をチューンしたり、別の環境に適用するには、jconf ファイルなどで Julius のオプションを指定する必要がある。そのためには、Julius の動作原理、さらには音声認識の基本原理の理解が必要であるので、以下に簡単に解説する。

5.1. 音声認識の基本原理

図5に連続音声認識の基本原則を示す。ここで、音響モデルは音素(ローマ字1文字にはほぼ相当)や音節(かな1文字に相当)の周波数パターンを保持し、入力音声とマッチングを行うものである。単語辞書は認識対象の語彙(=単語の集合)とその発音を規定し、ここで規定されているもののみがマッチングの対象となる。文字認識と異なり、文字を認識してから単語を照合するのではなく、単語辞書を照合しながら文字を認識することに注意されたい。これは、音声の本質的に続け字のように文字の区切りが明確でないためである。言語モデルは、単語の連鎖を規定するもので、決定的な記述文法を用いる場合は、それで規定されるもののみが照合の対象となる。

(しばしば誤解されるので)正確にいうと、Juliusは図5においてデコーダと呼ばれているモジュールであり、これは、与えられた音響モデルや言語モデルを用いて、入力音声 X を単語列 W に認識(デコード)するものである。商用のソフトウェアでは(内部的にはともかく)音響モデルや言語モデルが一体化されているのに対して、Juliusは音響モデルや言語モデルと完全に独立で、それらとのインタフェースもオープンなのが特徴である。したがって、これらを差し替えることにより、様々な入力環境やタスク、しいては日本語以外にも(あるいは音声以外にも?)適用することができる。

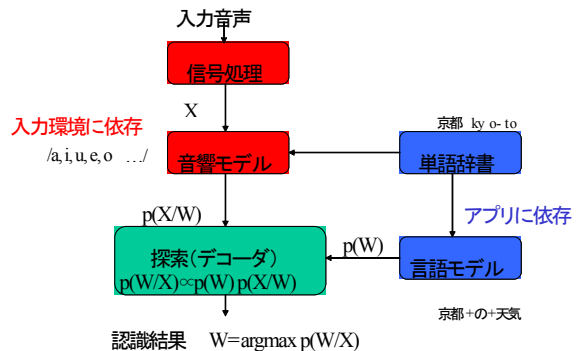


図5 音声認識の基本原則

次にやや専門的になるが、図5に基づく連続音声認識の原理を確率的な枠組みで説明する。音声認識は、入力音声Xに対する事後確率 $p(W|X)$ が最大となる単語列Wを見つける問題として定式化できる。事後確率 $p(W|X)$ を直接計算することは非常に困難であるので、ベイズ則により以下のように書き換える。

$$p(W|X) = \frac{p(W) * p(X|W)}{p(X)} \dots(1)$$

この分母については、Wの決定に影響しない正規化係数と考えられるので、これを無視することができる。分子については単純なパターン認識(数字の認識など)においては、Wの事前確率 $p(W)$ を等しいと仮定することができ、その場合は $p(X|W)$ のみで決定されることになるが、連続音声認識においては $p(W)$ も大きく関与する。

5.2. 音声認識における言語モデル

式(1)の $p(W)$ は(音声Xが入力される時点で)ある単語列Wのパターンが生起する確率であり、これは(音声Xとは無関係の)言語的な確からしさである。ディクテーションでは、日本語で使用される単語の統計量や、「私」の次には「は」や「の」がきやすいといった統計量に基づいて確率を推定している。タスクが限定されている場合は、文法的・意味的に正しくないものを除外する(確率0とする)ことにより、認識の候補を絞り込む。

単語辞書が語彙(認識対象の単語集合)を規定するのに対して、言語モデルは単語間の接続関係を規定するものであり、単純な単語音声認識はこれがない場合とみなすことができる。言語モデルは、統計的なモデルに基づくものと、決定的な記述文法に基づくものに大別できる。言語モデルの適用は、通常先頭の単語から逐次的に行われ、単語列 $W=\{w_1, w_2, \dots, w_N\}$ (w_i は各単語)に対して、

$$p(W) = \prod_i p(w_i | w_1 \dots w_{i-1}) \dots(2)$$

のようになる。これは、記述文法の場合は、単語列が受理されるか否か(1/0)で判定するオートマトン/パーザにより実現されるが、統計モデルの場合は、あまり履歴(確率の条件部分)が長いと組み合わせが膨大になり統計量の推定が困難になるので、 $p(w_i | w_1 \dots w_{i-1})$ を直近のN単語連鎖 $p(w_i | w_{i-N+1} \dots w_{i-1})$ で近似して用いることが一般的である。これを単語N-gramモデルと呼び、N=2(2単語連鎖)の場合がバイグラム、N=3(3単語連鎖)の場合がトライグラムである。

単語辞書と言語モデルは通常アプリケーションによって規定され、例えば天気案内やホテル検索などのシステムではそれらに特化したものを用意する。汎用的なディクテーション用のモデルは、日本語の大規模なテキストデータベースから構築しているので、それを例えばホテル検索システムにそのまま用いても、固有名詞がカバー

されなかったり、言語表現の予測能力が十分でないためにあまり高い性能は期待できない。

文法の記述が難しいような複雑度の大きいタスクにおいては、専用のN-gramモデルを用意するのが望ましい。N-gramモデルを学習するためのツールキットとして、CMU-Cambridge 統計言語モデルツールキット(<http://mi.eng.cam.ac.uk/~pre14/toolkit.html>) や palmkit (<http://palmkit.sourceforge.net>) がある。ただし学習に際しては、アプリケーションで想定されている(あるいは実際に発話された)文のテキストを大量に(数千文以上)収集する必要がある。

5.3. 音声認識における音響モデル

これに対して、式(1)の $p(X|W)$ は単語列Wから音声のパターンXが生起する確率であり、音響的なモデルによるマッチングに基づいて評価が行われる。こちらが通常のパターン認識処理に相当し、パターンの分布を推定したモデルを用いて行われるが、音声認識では時系列を柔軟に扱えるHMM(Hidden Markov Model)が主流になっている。

また、このモデル(テンプレート)の単位としては、音素(ローマ字1文字にほぼ相当)を用いることが多い。この単語と音素表記(もしくはかな表記)の対応づけは単語辞書で記述する。ここで音素表記は、できるだけ実際の発音に忠実であることが必要である。例えば、「京都」は正書法では「きょうと(kyouto)」と書かれるが、通常は「きょと(ky o- t o)」のように発声されるので、そのように記述する。格助詞「は」「へ」のようになんと読みが一致しない場合も要注意である。「日本(にっぽん/にほん)」のように複数の読みを持つ場合は、各々を記述する。

このようにして、単語列 $W=\{w_1, w_2, \dots, w_N\}$ が音素列 $\{m_1, m_2, \dots, m_N\}$ に展開されるので、 $p(X|W)$ は以下のように計算できる。

$$p(X|W) = \prod_i p(x | m_i) \dots(3)$$

ここで $p(x|m_i)$ は、通常音素単位の音響的特徴を表現したHMMを入力音声(の一部)xとマッチングすることにより計算する。

音素は連続的に発声されるので、各音素の音響的特徴が前後の音素によって大きく変動する。そのため、前後の音素に応じて別のテンプレートを用意するのがトライフォンモデルである。例えば、先行母音が*/i/*で後続の母音が*/a/*の場合の子音*/k/*は*i-k+a*のように表記される。ただしこれは、*i-k+a*の三つ組全体に対するテンプレートではなく、あくまで子音*/k/*に対するテンプレートである。したがって、「会社(かいしゃ)」という単語に対するトライフォンによる表記は、*[k+a k+a+i a+i+sh ish+a sh-a]*のようになる。トライフォンに対して、前後の音

素に依存しないモデルをモノフォンと呼ぶ。

音響的特徴は、話者（性別・年齢層など）や入力環境（パソコンの接話マイク・電話・自動車内など）によっても大きく影響されるので、利用する条件に適した音響モデルを使用する必要がある。話者については、不特定話者のモデルを利用することもできるが、小児や高齢者は特別なモデルが必要である。入力環境については、ミスマッチが認識性能に重大な影響を与えるので、例えばパソコンと電話と自動車内では全く異なる音響モデルが必要である。ここで挙げた典型的なものについては、連続音声認識コンソーシアム (<http://www.lang.astem.or.jp/CSRC/>) の成果物として有償で入手可能である。

自力で HMM を学習するためのツールキットとして、HTK (<http://htk.eng.cam.ac.uk/>) があるが、音響モデルの構築には、言語モデル以上に学習データの収集が大変であり、またかなりの専門的知識も必要とする。

5.4. デコーダ Julius の動作原理

音声認識の過程においては、式(1)を様々な単語列 W の仮説について計算し、その中で最も事後確率の高いものを選択する (= 最大事後確率原理)。前述のように分子のみを考慮し、対数スケールに変換すると以下のように書き換えることができる。

$$\begin{aligned} \hat{W} &= \arg \max p(W | X) \\ &= \arg \max p(W) * p(X | W) \quad \dots(4) \\ &= \arg \max \{ \log p(W) + \log p(X | W) \} \end{aligned}$$

実際には、式(4)の $\arg \max$ の中身の評価関数を以下のように設定することが一般的である。

$$\log p(X | W) + \alpha \log p(W) + \beta * N \dots(5)$$

ここで、 α は言語モデル重み、 β は単語挿入ペナルティと呼ばれるパラメータであり、 N は仮説 W に含まれる単語数（単語列の長さ）である。

単語列 W の仮説数は膨大になるので、この評価値 (= 尤度) の計算を先頭から逐次的に行い、尤度が小さい候補は途中で除外（枝刈り）し、尤度が大きい候補についてのみ次の単語を接続・展開する。この候補を残す範囲（ある時点における仮説数）をビーム幅と呼ぶ。このようにモデルによる予測・尤度計算を行いながら、最尤仮説の探索を行うプログラムが Julius である。

Julius は 2 パスの探索アルゴリズムを採用している。すなわち、第 1 パスで単語バイグラムモデルを用いて荒い照合を行い、その中間結果に対して第 2 パスで単語トライグラムモデルを適用して、最終的な認識結果を求める。また音響モデルについても、第 1 パスでは単語間についてはトライフォンを厳密に適用せず、候補を絞った第 2 パスにおいて正確な尤度を計算する。これにより、

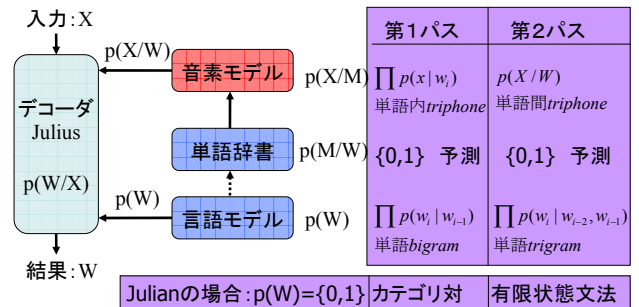


図 6 Julius の動作原理

全体としての処理時間と使用メモリ量の効率化を図っている。第 1 パスは入力に対してほぼリアルタイムに処理が進み、いったんその結果を出力するが、実際には第 2 パスにおいて候補を再評価し、最終的な認識結果を確定する。

この Julius の動作原理を図 6 にまとめる。図 5 における信号処理のモジュールも Julius に含まれている。

6. Julius の主要な設定手順

本章では、Julius による音声認識システムの構成における主なポイントについて述べる。

6.1. 入力の指定

入力として、マイクなどのオーディオデバイス以外に、音声ファイル（wav 形式）、音響特徴量ファイル（HTK 形式の MFCC ファイル）を指定することができる。信号処理については、MFCC（メル周波数ケプストラム係数）とパワー、及びそれらの差分の計算しか実装されていない。他の音響特徴量を用いる場合は、その信号処理モジュールとそれで学習された音響モデルを用意する。

6.2. 言語モデルの選択・指定 --統計モデル(N-gram)か? 記述文法か?--

言語モデルは原則として、アプリケーション毎に用意する。

タスクが単純で明確な場合は、4 章で説明したように、人手で文法を記述する。この場合、Julian を使用し、用意した文法と単語辞書のファイルを実行時に指定する。

汎用的なディクテーションでは、3 章のようにディクテーション実行キットに含まれるトライグラムモデルを使用する。ただし、このトライグラムは新聞記事などの書き言葉で学習されているので、話し言葉に対しては十分に対応できないことに注意されたい。

5.2 節の最後で述べたように、タスクやドメインに応じて、N-gram モデルを用意する場合は、前向きバイグラムと逆向きトライグラムモデル、及び単語辞書のファ

イルを指定する。

6.3. 音響モデルの指定

音響モデルは使用環境によって決定する。できるだけ使用時と同一の条件（帯域・入力装置・周囲騒音条件等）で学習されたモデルを用いるようにする。

Julius のディクテーション実行キットでは、16kHz サンプリングの接話マイクを想定した成人用音声モデルを用いている。その他、5.3 節で述べたように、小児や高齢者などの話者層、電話や自動車内などの使用環境に応じて適切なモデルを用意・選択する。

トライフォンモデルの場合は、マップファイルと呼ばれる単語辞書上で出現するエントリと実際に音響モデルで用意されたエントリとの対応表も指定する（これは通常音響モデルに添付されている）。

電話音声のようにサンプリング周波数が異なる場合は、信号処理のオプションも変更する必要がある。

6.4. 認識時（デコーディング）パラメータの指定

ビーム幅や言語モデル重み・単語挿入ペナルティなどの指定を行う。これらは言語モデルの確率 $p(W)$ が求められる統計言語モデルのときに有効である。これらのパラメータの値はデフォルトでも設定されているが、チューンすれば、認識精度や処理速度が多少改善される。

6.5. 出力時（デコーディング）パラメータの指定

認識結果について、いくつまで(N-best)候補を求めらるか、音素列や単語境界などの情報も出力するか、などの指定を行う。デフォルトでは第一候補の単語列のみを出力するが、"-n", "-output" で指定した数の候補を求めた上で出力できる。また "-walign", "-palign", "-salign" で、それぞれ単語単位、音素単位、状態単位の境界情報を、"-progot" で第 1 パスの途中結果を出力できる。"-quiet" を指定すると、単語列以外の出力を抑制する。

6.6. jconf 設定ファイル

以上で説明したようなモデル指定やパラメータ指定などの設定は、実行コマンド("julius"または"julian")の引数で指定してもよいが、非常に多くなるので、「jconf ファイル」と呼ばれる設定ファイルに記述すると便利である。

音響モデル(トライフォンモデルのマップファイルも)や言語モデル・単語辞書の指定に加えて、ビーム幅・言語モデル重み・単語挿入ペナルティなどのデコーディングのパラメータの指定、さらに音声入力ソースの選択や出力内容の指定など、全ての設定がこのファイルに記述できる。jconf ファイルは単純なテキスト形式である。図 7 に jconf ファイルの例を示す。詳細はディクテーション実行キットに含まれる標準設定の "fast.jconf" と軽量動作設定の "light.jconf"、及び Julius のアーカイブに付

```
-d model/75.20k.1-1.10p.wit.bigram.gz          % 言語モデルの指定
-v model/20k.htkdic.gz                          % 単語辞書の指定
-h model/hmmdefs_ptm_gid.binhmm               % 音響モデルの指定
-hlist model/logicalTriList                    % トライフォンマップファイル

-lmp 8.0-2.0-lmp2 8.0-2.0                      % 言語モデル重みと単語挿入ペナルティ
-b 600                                          % ビーム幅 (第1パス)
-s 500-m 2000-wb 30                           % ビーム幅 (第2パス)
-n 1                                           % 出力候補(N-best)数
-demo                                          % デモ用簡易出力
-input mic                                     % マイク入力
```

図 7 jconf ファイルの例

属のマニュアルを参照されたい。

Julius 実行時には、使用する jconf ファイルを "-C" で指定する。jconf ファイルは複数指定することもでき、ある jconf ファイルで別の jconf ファイルを参照・指定することもできる。さらに、jconf ファイルとコマンド引数指定を併用することもでき、その場合後者が優先される。

6.7. Julius カスタマイズサイト

上記のような Julius の設定を、メニュー画面を用いて行えるようにするサイトを用意し、Julius の Web サイトからリンクする予定である。

6.8. Julius サーバーモード

Unix 版は基本的にスタンドアローンで動作するが、"-module" を指定することでネットワーク認識サーバーとして動作させることも可能である。このとき、Julius/Julian は他のアプリケーションからの接続を待ち、接続後にクライアントとの間で、音声の受信や認識結果の送信を行う。また、クライアントからの認識の中断や再開の制御、文法の動的切り替えなども行える。外部アプリケーションから音声認識機能を利用したい場合などに有用である。なお、サンプルのクライアント "jcontrol" が Julius に付属している。図 8 にクライアントに送られる情報の例を示す。

```
<INPUT STATUS="LISTEN" TIME="1084265467"/>
<INPUT STATUS="STARTREC" TIME="1084265467"/>
<INPUT STATUS="ENDREC" TIME="1084265467"/>
<INPUTPARAM FRAMES="233" MSEC="2330"/>
<RECOGOUT>
<SHYPO RANK="1" SCORE="6065.173340">
  <WHYPO WORD="" CLASSID="" PHONE="silB" CM="0.445"/>
  <WHYPO WORD="年金" CLASSID="年金+ネンキン+2" PHONE="neNkiN" CM="0.579"/>
  <WHYPO WORD="改葬" CLASSID="改葬+カイカク+17" PHONE="kai kaku" CM="0.927"/>
  <WHYPO WORD="法案" CLASSID="法案+ホーアン+2" PHONE="ho a N" CM="0.026"/>
  <WHYPO WORD="." CLASSID="." PHONE="sp" CM="0.664"/>
  <WHYPO WORD="" CLASSID="" PHONE="silE" CM="1.000"/>
</SHYPO>
</RECOGOUT>
<INPUT STATUS="LISTEN" TIME="1084265467"/>
```

図 8 認識サーバーから送信される情報の例


7. Windows SAPI 版の使用法

本章では、Windows の Speech API(SAPI)に対応した Julius SAPI 版について説明する。

なお、Unix 版を Windows 上へ移植した「Windows コンソール版」もあるが、こちらの使い方や jconf ファイルの記述方法は Unix 版と全く同じである。

SAPI はマイクロソフト社が策定した API で、SAPI-5 が Windows XP に標準で含まれている。XP より前の Windows ではマイクロソフト社の Web サイト (<http://www.microsoft.com/speech>) からダウンロード・インストールする必要がある。なお、SAPI には以前 SAPI-4 があったが、SAPI-4 と SAPI-5 では設計思想が基本的に異なり、互換性もないので注意されたい。Julius SAPI 版は SAPI-5 に対応したものである。

Julius SAPI 版を使用するに先立って、Julius 本体 (2004 年 10 月時点で ver.2.3) と標準モデル一式を、Julius の Web サイト (<http://julius.sourceforge.jp/sapi>) からダウンロード・インストールする。

Windows の「コントロールパネル」の「音声認識」のプロパティをクリックすると、「音声認識」の「言語」において、「Julius for SAPI Engine」が選択されているはずである (図 9)。

音声認識を起動後に、マイクを接続して発声したときに、図 9 の下の「マイク」のインジケータが適度に振れるのが正常な状態である。ここで問題があれば、「マイク入力」のところで音量レベルや入力デバイスの設定を変更する。できれば事前に、Windows 付属のサウンドレコーダなどで音声が見事に録音できていることを確認することが望ましい。マイク自身や接続の確認も含めて、実行環境における録音の音質には特に注意が必要である。

図 9 のプロパティの「設定」ボタンをクリックすることにより、6 章で述べたような設定ができる。

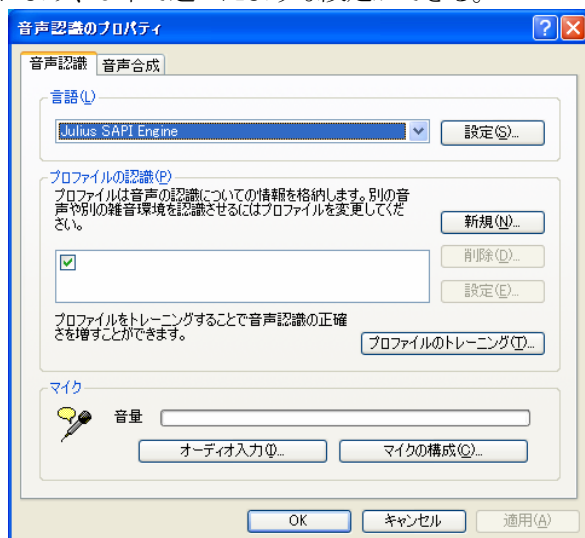


図 9 「音声認識」のプロパティ

具体的には、「音響モデル」のプロパティで、音響モデルとトライフォンのマップファイルを指定する。標準では接話マイクを想定した成人用音声モデルが指定されている。電話音声などでは専用のモデルを用意するとともに、「音響処理」のプロパティで信号処理の設定も変更する。

「言語モデル」のプロパティで、統計モデルと単語辞書ファイルを指定する。記述文法の場合は、実行時に SAPI を通して文法ファイル (単語辞書情報も一体化した XML ファイル) を指定する。図 10 に果物注文タスクの文法 (図 3 のものとほぼ等価) の例を示す。

```
<?xml version="1.0" encoding="UTF-8" ?>
<GRAMMAR>
  <RULE name="S" toplevel="ACTIVE">
    <RULEREF name="fruit_n" />
    <O>
      <RULEREF name="num" />
    </O>
    <RULEREF name="please" />
  </RULE>
  <RULE name="fruit_n" toplevel="INACTIVE">
    <L proptype="fruit_n">
      <P valstr="蜜柑"/><P valstr="蜜柑/みかん"/>
      <P valstr="リンゴ"/><P valstr="リンゴ/りんご"/>
      <P valstr="ぶどう"/><P valstr="ぶどう/ぶどう"/>
    </L>
  </RULE>
  <RULE name="num" toplevel="INACTIVE">
    <L proptype="num">
      <P valstr="0"/><P valstr="0/ゼロ"/>
      <P valstr="1"/><P valstr="1/いち"/>
      <P valstr="2"/><P valstr="2/に"/>
      <P valstr="3"/><P valstr="3/さん"/>
      <P valstr="4"/><P valstr="4/し"/>
      <P valstr="5"/><P valstr="5/ご"/>
      <P valstr="6"/><P valstr="6/ろく"/>
      <P valstr="7"/><P valstr="7/なな"/>
      <P valstr="8"/><P valstr="8/はち"/>
      <P valstr="9"/><P valstr="9/きゅう"/>
    </L>
  </RULE>
  <RULE name="please" toplevel="INACTIVE">
    <L>
      <P>を</P>
      <P>にして</P>
    </L>
    <P>ください</P>
  </RULE>
</GRAMMAR>
```

図 10 SAPI の XML 文法の例 (果物注文タスク)



図 11 「デコード」のプロパティ

「デコード」のプロパティ (図 11) では、第1パス、第2パスそれぞれのビーム幅、及び言語モデル重み・単語挿入ペナルティを指定できる。また、N-best 候補数、信頼度計算の指定 (N-best 指定が必要)、認識失敗判定、単語境界の計算なども指定できる。

これらを試すことができるサンプルプログラムや典型的なタスクのサンプル文法が Web サイトにあり、Javascript で書かれた reco.html などがわかりやすいと思われる。

8. うまくいかない場合は

比較的静かな環境で簡単な文を明瞭に発声した場合は、通常 80~90%以上の認識率が得られると期待される。もちろん音声認識はまだ不完全な技術で、多くの限界があるが、あまりに動作がおかしい場合は設定に問題があると考えられる。具体的には、数百単語以下で文法を記述した場合や、ディクテーションで「マイクのテスト」や「衆院議員は具体的にどう考えているのか」などと発話した場合に、認識がうまくいかない場合は、何らかの問題がある可能性が高い。以下に典型的なトラブルを列挙する。

8.1. マイク・オーディオデバイスの問題

マイクやオーディオデバイスから音声が入力されていない、これらの品質が著しく悪いためにノイズが重畳している、ボリュームレベルが大きすぎる (音が割れている) / 小さすぎる (音が検出されない)、といったことがないか確認する。実際に音声を録音してみて、ヘッドフォンで確認するのが望ましい。Julius には認識処理を行った音声をファイルに保存する機能もあるので、これを

利用するとよい。文字認識と比べて、このような入力時の問題をユーザが感知しにくいと思われる。

また、サウンドデバイスが 16kHz, 16bit, モノラルで録音できるかについても念のため確認する。音声 (WAV) ファイルを入力とする場合は、上記の条件に加えて無圧縮のフォーマットであることを確認する。

8.2. 音響モデルのミスマッチ

標準の音響モデルは、比較的静かな環境で、成人話者が発声することを前提としている。5.3 節で述べたように、小児や高齢者などの話者層、電話や自動車内などの入力環境に対しては、専用の音響モデルが必要である。これらは、連続音声認識コンソーシアム (<http://www.lang.astem.or.jp/CSRC/>) の成果物として有償で入手可能である。

8.3. 文法・単語辞書の問題

文法を記述した場合に、想定される文が受理されるようになっていない、逆に文法の制約が弱く非文法的な文をあまりに多く受理してしまう、などの問題が生じる。Julius のパッケージには、文法で受理できるかチェックしたり、ランダムに文を生成するツールが添付されているので、音声認識の前にこれらで確認する。

細かいことであるが、文頭・文末にポーズと、できればフィルター (「えーと」など) を入れておく必要があり、また文中にもポーズが置かれる可能性がある箇所には NOISE(NS)を入れておく。

また、単語辞書で発音が実際のものに近い形で記述されているか、助詞の「は」「へ」などの読みが正しく与えられているか、なども注意する。

8.4. 技術的境界

現在の音声認識技術は、例えていうと外国語話者の上級レベルと考えられ、我々が外国語を聞き取る際に直面する以下のような問題には対処できない。ただし、タスクを限定して、語彙や文法の制約を強くすれば、ある程度動作させることができる。

- 話し言葉

話者が機械による音声認識を意識していない状況、例えば、講演や会議・会話などにおいては、口語的な表現が多く、また個々の発音が明瞭でないので、音声認識は非常に困難になる。ディクテーション実行キットに含まれているモデルはこのような話し言葉に対応していない。この対応については、現在 (著者らにより) 精力的に研究・開発が進められている。

- 早口・ささやき声・なまり・感情的な音声

これらには標準の音響モデルでは対応できない。

- 雑音

雑音の要因として、発話している周囲の騒音から、

パソコンのファンの音など入力機器の雑音、スピーカから出ている音の回り込みなどが含まれる。録音した音声ヘッドフォンで聞いて雑音が耳につくレベル (SN 比が 20dB 以下) になると、著しく認識精度が低下する。雑音下での音声認識の研究も継続的に行われているが、大語彙連続音声認識で認識精度を確保するには接話型マイクの使用が望ましい。

- 遠隔発話・集音マイクによる入力

マイクから距離が離れると、周囲の騒音レベルは小さくても、SN 比は著しく低下するし、残響 (反射音) の影響も受けるので、認識精度が著しく低下する。遠隔発話に対する研究も進められているが、発声環境に応じた特別な信号処理が必要となる。

9. おわりに

音声認識技術は 1990 年代になって、HMM に代表される統計的モデルの確立、その学習を可能にした大規模コーパスの集積、そして実時間認識を可能にした計算機能力の向上に伴って、大きな進歩を遂げた。その反面音声認識システムが大規模になりすぎて、個々の研究者・研究機関ですべてを開発・維持していくのが難しくなった。Julius は、そのような背景のもとで、研究者を主な対象としてオープンソースの音声認識ソフトウェアとして開発された。実際に国内の数多くの研究機関で、事実上のベースライン/リファレンスとして利用されている。

また、ヒューマンインタフェース関連を中心に、様々なアプリケーションシステムの研究・開発においても利用されている。

しかしながら、(Julius に限らず) まだ世間一般で普遍的に音声認識が利用されていないのも事実である。音声は人間の最も自然なコミュニケーション手段と考えられるが、それ故に (人間と同程度に) 自然なレベルで使えるようにならないと、結局のところ他のモダリティ (GUI やテンキー) に勝てないという見方もある。もちろん、ヒューマンインタフェースの問題も大きい。

いずれにせよ、音声認識の性能が母国語話者と同程度になるまでにはまだかなりの研究を要するのは間違いなく、その過程において Julius が貢献できれば幸甚である。

謝辞

本ソフトウェアの研究開発に際して多大な協力を頂いた方々、特に鹿野清宏教授 (奈良先端大)、武田一哉教授 (名大)、伊藤克亘助教授 (名大) の各氏に深く感謝する。また現在の開発は、文部科学省リーディングプロジェクト「e-Society 基盤ソフトウェアの総合開発」の下で行われている。

参考文献

[鹿野 01] 鹿野清宏, 伊藤克亘, 河原達也, 武田一哉, 山本幹雄 編著: 「音声認識システム」, オーム社, (2001) (ISBN 4-274-13228-5).

[河原 00] 河原達也. ここまできた音声認識技術. 情報処理, Vol.41, No.4, pp.436-439, (2000).

[河原 04] 河原達也. 話し言葉による音声対話システム. 情報処理, Vol.45, No.10, pp.1027-1031, (2004).

著者紹介

河原達也 (正会員)

1987 年 京都大学工学部情報工学科卒業。1989 年 同大学院修士課程修了。1990 年 同博士後期課程退学。同年 京都大学工学部助手。1995 年 同助教授。1998 年 同大学情報学研究科助教授。2003 年 同大学学術情報メディアセンター教授。現在に至る。この間、1995 年から 1996 年まで 米国ベル研究所客員研究員。1998 年から ATR 客員研究員。1999 年から 2004 年まで 国立国語研究所非常勤研究員。2001 年から 2004 年まで 科学技術振興事業団 さきがけ研究 21 研究者。音声認識・理解の研究に従事。京大博士 (工学)。1997 年度 日本音響学会栗屋賞受賞。2000 年度 情報処理学会坂井記念特別賞受賞。情報処理学会連続音声認識コンソーシアム代表。IEEE SPS Speech TC 委員。情報処理学会、電子情報通信学会、日本音響学会、言語処理学会、IEEE 各会員。

李 晃伸 (非会員)

1996 年 京都大学工学部情報工学科卒業。1998 年 同大学院修士課程修了。2000 年 同大学院情報学研究科博士課程短期修了。同年 奈良先端科学技術大学院大学情報科学研究科助手。現在に至る。主として音声認識アルゴリズム・音声言語理解の研究に従事。京大博士 (情報学)。2001 年度 日本音響学会栗屋潔学術奨励賞受賞。情報処理学会、電子情報通信学会各会員、日本音響学会、IEEE 各会員。