Vol.2018-NL-237 No.5
2018/9/26

音声対話システム向け意味属性抽出と意図タイプ推定実装小型化

米持幸寿[†1]

概要：音声対話システムの構築において，自然言語理解(Natural Language Understanding: NLU)のための意味属性抽出と意図タイプの推定は基本的な自然言語処理である．過去の対話システム研究において，日本語における形態素解析，品詞推定，係り受け解析，パターンマッチングなどを，様々な OSS を組み合わせることで実現している例が多く存在する．しかし，そういったシステムは複雑かつコード量が多いという課題も存在する．そのような特徴はオフライン小型ロボットの組み込み用途に使う場合には障壁となる．本研究では，プログラミング言語が標準で装備している正規表現のみを活用することでコード量を劇的に削減した上で同等の機能を実現する実装を試作した結果を紹介する．

キーワード：音声対話システム，意味属性抽出，文型マッチング，正規表現

1. はじめに

音声認識技術の高性能化が後押しとなって音声対話型のシステムが普及し始めている．音声対話アプリケーションを実装するにあたり言語処理は欠かせない．商品化されている技術の多くが固有表現抽出[1]を含む特定の単語の抽出(意味属性抽出)とユーザーが何を意図したかの推定(意図タイプの推定)によりプログラムの振り分けを行う機能を提供しており，クラウドでの機能提供をしている．クラウドを使う一つの理由は豊富な計算資源が利用できることである．なぜならば, FST などを使って音声認識の間違えを補正する仕組みを導入したり(須藤ら)[2], (勝丸ら)[3], プロトタイピングを容易にしたり(福林ら)[4]することで，精度を上げることに重きを置いてきたからである．しかしながら,精度を落としてでも，ある程度オフラインで処理ができるプログラムを小型のロボットやスマートフォンに実装したいというニーズも依然存在している．音声認識の信頼度が向上したことでその部分の処理を簡素化できる可能性もある．

従来の実装方法ではフットプリントが大きいか，クラウドに機能を置くオンライン型としているものが多く，小型のロボット上でオフライン動作させるニーズを満たすことができない．本研究では, OSS を利用しないことでフットプリントを小さくしてエッジ側で実装可能とし，オフラインでも動作する言語処理を実装することを目的としている．

まず従来の実装方法の状況を説明し，言語処理機能として正規表現を採用するにあたり，機能性と性能に関する工夫点を説明する．さらに懸念される速度性能の評価を示す．

2. OSS を多用する従来方法

従来の方法では，様々な言語処理を基本としており，例えば以下のような構成の OSS を利用してルールベースによる言語処理を構成することが考えられる．

- 日本語用に lucene-gosen[5]または MeCab[6]を利用した形態素解析と品詞推定．
- 英語用に同目的で Apache OpenNLP[7]を利用．
- OpenFST[8](WFST によるパターンマッチング処理)により意図推定．

これらを利用した上で，以下のような処理を行う．

1. ユーザー発話意図に紐付けられた，意味属性の出現箇所(スロット)を指定したユーザー発話パターンを準備しておき, FST モデル(A)を生成しておく．
2. 入力されたテキストを単語単位に分解する．
3. 単語の組み合わせから辞書引きを行い，意味属性候補を抽出する．
4. 抽出された意味属性をスロット候補とした入力文字用の FST モデル(B)を生成する．
5. FST モデル(A)と(B)とを WFST を使ってマッチングを行い，ユーザー発話意図がどれだったか推定する．

意味属性抽出は，場合によっては複数の組み合わせの可能性が示されることがある．これに対して WFST 処理を行うことで尤も確からしいものを選択することができる．これが，この推定方式の特徴でもある．

一旦，従来の実装方法において最低限必要なものだけを集めた場合，前提 OSS を含め 370M バイトであった．スマートフォン，組み込みデバイス，小型ロボットなどではメモリーやストレージに制約が多く，このサイズはまだ十分大きい．より小さいデバイスに対応するためのソフトウェア小型化の試作を本研究でさらに行なった．

3. 正規表現による推定の可能性検討

小型化するにあたり，従来の実装方法のフットプリントの大半を占めている OSS の排除をまず検討した．まず，形態素解析を行っているものの, FST モデル化するのみで品詞情報を使用していないため NLP モジュール(lucene-gosen, MeCab, OpenNLP)は不要と考えた．次に FST は，文字列の並び順のマッチングを行なっているため正規表現処理と類似していると仮定し, FST を使わずに正規表現処理の適用を検討した．

従来の実装方法では形態素解析によって分解された単語

†1 (株)ホンダ・リサーチ・インスティチュート・ジャパン
Honda Research Institute Japan, Co. Ltd.

を並べて FST モデルとなる有向グラフを構築している．FST モデルの単語は単純比較される．単語は言い回しなどによって変化する可能性がある．いわゆる「ゆらぎ」である．これらのことから，比較用 FST モデルをあらかじめ用意するとき，ことばのゆらぎの可能性を幅広く網羅的に準備する必要がある(例 1)．

例 1　移動時間を訪ねる文

<駅名>へ何分
<駅名>に何分
<駅名>へは何分
<駅名>には何分
<駅名>まで何分
<駅名>まで何分で行ける
<駅名>まで何分かかる
<駅名>まで何時間
<駅名>まで何時間で行ける
<駅名>まで何時間かかる
<駅名>まで何時間かかるか教えて

この文型にマッチする入力テキストが来たとき，これらをすべて「ある場所の時刻を尋ねている」と推定できればよい．また，これらに該当しないものを排除できればよい．これを概念的な有向グラフで表すと図 1 のようになる．

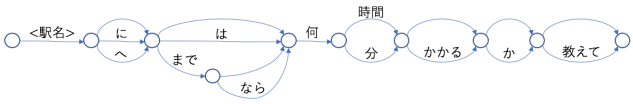

図 1 有向グラフの例

FST では，他の意図を示すユーザー発話パターンを含めすべてルートとしてもつ状態モデルを一つ作ることになるが，本論では説明を省略する．

これらを正規表現で表すと，例 2 のようになる．

例 2　正規表現での記述

<駅名>(に|へ)*(は|まで(なら)*)*何(時間|分)(かかる)*(か)*(教えて)*

これら二つの例から推測するに，このパターンマッチングには FST のような有効グラフは必須ではなく，文字列の比較パターンである正規表現で十分と考えた．正規表現は世界的にもよく知られている記法であり,アプリケーション開発においても識者が多いという点において有意義である．また，多くのプログラミング言語が機能を提供しており，実装する上でフットプリントを小さくすることにおいて有効である．そこで本研究では，例 2 のような正規表現記述を用いて意味属性の候補抽出と意図推定を行える小型のモジュールを試作することを目標とした．

4. 制約

ロボットが特定のシナリオで音声対話を行うに伴い必要となる機能を維持しつつ，小型の実装を作れるようにするため，以下の制約を持って設計を行った．

- 辞書を利用して意味属性候補抽出ができること．
- 複数の辞書単語に該当する可能性があるとき，ユーザー発話パターンマッチングを組み合わせることで，より正しい意味属性の組み合わせと意図を推定できること．
- 可能な限り OSS を利用せず，プログラミング言語が標準で備える機能のみで実装してみること．

ただし，以下の制約は逆に排除することとする．

- 大量辞書に関する処理は実装しない．辞書単語数を巨大化するための技術は大小様々なものがデータベースとして既存しており，辞書引きの箇所に簡単に組み込むことができる．
- ユーザー発話パターンの処理の重さを気にしない．小型化が目的であり，ユーザー発話パターン数が巨大なものをスコープとしない．従来の実装方法が大規模の為のものであり，今回のものは小型化が目的である．また，大型のものはクラウドに既存している．オンプレミスで独自に持つシステムの場合，人手で定義を作るユーザー発話パターンがパフォーマンスに影響が出るほど大きくなることを予想できない．

5. 提案する手法

従来の実装方法では，WFST を使ってユーザー発話パターンの一致確率を計算している．このため，たくさんのパターンから選択するための処理は一度の処理である．今回の実装では，パターンの記述そのものが正規表現であり，パターンごとに正規表現のマッチング処理を行う必要がある．このため，パターン数が増えてしまうと正規表現処理回数が増えてしまうという懸念がある．これの回避方法，すなわち正規表現処理回数削減方法をここで説明する．

5.1 オントロジー辞書

ここでは表 1 のような辞書単語がオントロジー辞書に登録されていると仮定する．

表 1　オントロジー辞書

型	id	言葉
station	tokyo	東京, Tokyo
	shibuya	渋谷, Shibuya

5.2 ユーザー発話パターンのスロットの抽象化

ユーザー発話パターンには ＜＞ で挟まれたスロットが定義されている．パターンに設定されるスロットにはスロ

ット名と型が定義される．これは，同じ型の意味属性を二つ以上同時に取得することが考えられるためである．たとえば，二つの駅を指定するような場合に，一つ目の駅名を s1，二つ目を s2 などと指定する(例 3).

例 3　二つの駅名を受信するパターン定義

<s1:station>駅*から<s2:station>駅*への乗り換え(を教えて|は)*

このままでは，後述する入力テキストとの比較をする際にスロット名が邪魔になるため，スロット名を除去する(例 4).

例 4　比較パターン

<station>駅*から<station>駅*への乗り換え(を教えて|は)*

さらに，このパターンがどういう型のスロットを持っていたかを生成して保持する(例 5).

例 5　スロット型比較パターン

station - station

5.3 抽出したい型の収集

意味属性をやみくもに探さないよう，ユーザー発話パターンに指定されているスロットの型を収集する．例 3 のユーザー発話パターンでは station 型が収集される．これは，膨大な辞書があったとき，入力文字列から，どの型の意味属性をスロット候補として収集すべきかの一覧となる．

ここまではシステムの起動処理である．

5.4 入力テキスト上の意味属性候補のスロットへの変換

音声認識器等からの入力文字列処理の最初のステップが意味属性候補の収集である．オントロジー辞書を使うラベル付けにおいて文字列を n-gram に分解し，辞書に登録されている単語と比較する.ここでは比較しやすいように二つの入力テキストと入力用検査文字列を例示する(例 6，例 7).

例 6　意味属性が一つの入力テキスト

入力(1)	東京駅にいます
検査文字列	<station>駅にいます

例 7　意味属性が二つの入力テキスト

入力(2)	東京駅から渋谷への乗り換えを教えて
検査文字列	<station>駅から<station>への乗り換えを教えて

前出のユーザー発話パターンにおいて station 型を収集する必要があるため，一致した文字列の上位語に station があるものがラベル付けされている．

ユーザー発話パターンと同様に，抽出された意味属性の型の一覧を生成する(例 8).

例 8　スロット型比較パターン

(1)	station
(2)	station – station

5.5 対象ユーザー発話パターンの選定と正規表現処理

ユーザー発話パターンから生成されたスロット比較パターン(例 5)と，入力テキストから抽出された意味属性から生成されたスロット型比較パターン(例 8)を比較することで，例 8-(1)の入力テキストは該当せず，例 8-(2)の入力テキストは該当し，すなわち抽出された意味属性の組み合わせが合致することがわかる．これは，(2)の入力テキストは，例 2 のパターンと比較する価値がある，ということを意味し，例 6-入力(1)のテキストを**例 4 の比較パターンで比較する必要はない**ということを意味する．正規表現比較処理数を削減でき パターン数が増大しても処理時間に影響が出ないようにしている．

結果的に，以下の意味属性が抽出され，パターン(2)に合致したという推定結果が出力される(例 9).

例 9　出力例

スロット名	値(id)	表記
s1	tokyo	東京
s2	shibuya	渋谷

6. 試作

試作するにあたり，以下の方法を選択した.

1. Java で実装する
2. 辞書データの保持と検索には Java の標準機能のコレクションクラス群(HashMap, ArrayList など)を利用し，インメモリーで管理する．入力テキストに対して先ず辞書引きによってスロットの位置候補を生成し，入力テキスト上で辞書単語にマッチした箇所をスロット候補文字列で置き換える．これを検査用入力文字列と呼ぶ．複数のスロット候補が生成された場合，検査用入力文字列は複数作られることがある．
3. ユーザー発話パターンの記述には正規表現を使う．ただし，スロット位置の指定には<s1:station>のようなパターンを新たに導入する．
4. ユーザー発話パターンごとに設定されているスロットの候補リストを保持する．

5. ユーザー発話パターンのマッチング処理には Java の標準機能の正規表現を使う.全ての組み合わせで正規表現処理は行わず, スロット候補の組み合わせが同じになるもののみを処理する. こうすることで, 不要な正規表現処理を削減でき, 辞書が大きくなったりユーザー発話パターンの数が増えたりした場合にパフォーマンスが著しく低下することをある程度防止することが可能.
6. 正規表現によってマッチングが陽性となったパターンリストと意味属性のセットを出力する.

試作によって開発された NLU モジュールは, 参考として用いた従来の実装方法の対話システムの言語処理部にて事前に作られたサンプルアプリケーションのすべてを実行でき, 機能性はある程度同等であることを示した. また, 実装は 27k バイトとなった. 従来の実装方法のおよそ 1/14,000 のサイズである.

7. 速度性能評価

今回の試作では, 意味属性を抽出するためのオントロジー辞書と, ユーザー発話意図を推定するための正規表現パターンが, 特殊な推定アルゴリズムを使わずとも, Java の基本機能のみでどれくらい実用的な性能で実現できるか, という検証を行っている.

スロットの組み合わせの比較により余分な正規表現処理を行わないことで, パターン数の増加に対してスケーラブルなはずである. それを証明するため, 多くのパターン数を入力する実験を行なった.

意味属性抽出をして文型の照合を行うことから, 辞書単語数と正規表現パターンの数とで処理速度が左右されることになる. また, それらのデータの増減はプログラムの初期化と検査処理の速度に影響が出ることから, その双方の測定を行なった.

論理的には大量の辞書を作ることは困難なので,「一二三四五」「一二三四六」というような文字列を大量に生成して単語として登録し利用した. 単純に順番に型を定義し, その該当スロットを持つパターン定義をランダムに生成し, それを入力文字列として使う.

また, 実験環境として以下の環境を利用している.

- Apple MacBook Early 2015 モデル
- プロセッサ：Intel Core M 1.2GHz
- メモリー：DDR3 8GB 1600GHz
- MacOS High Sierra 10.13.4
- Java 1.8.0_92

7.1 辞書単語数の影響

辞書引きによって地名や建物名などを意味属性として抽出しようとすると, どうしても辞書単語数が膨大になる. そういった場合を考慮して万単位の単語数に対してどのように時間がかかるか計測することとした.

今回の実装では, オントロジー辞書を CSV ファイルに準備し, Java の HashMap を活用したデータ構造で実現している. このため, この性能は, ファイルの読み込み速度と HashMap の性能そのものと考えることもできる. 測定結果を表 2 および図 2 に示す.

表 2 単語数の影響

単語数	読み込み時間(ms)	検査時間(ms)
0	0	0
5000	627	845
10000	4111	794
15000	13918	780

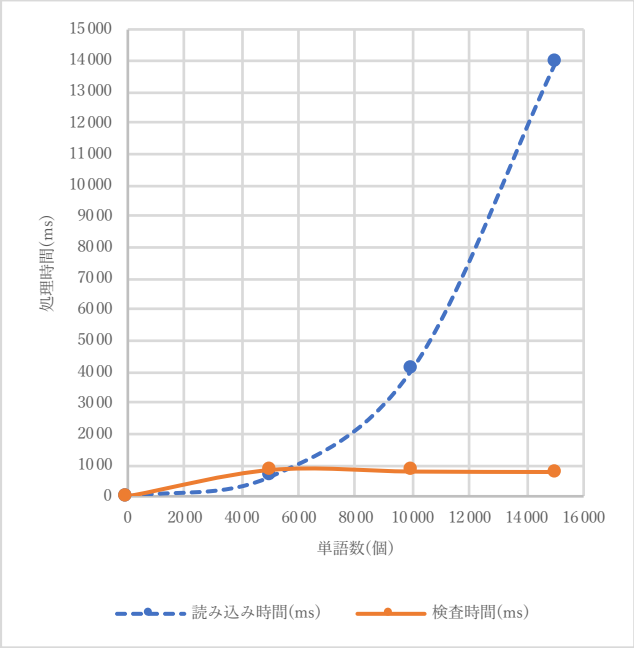

図 2 単語数の影響グラフ

読み込み時間はロボットシステムの起動速度に直接影響するため, あまり長くかかることは好ましくない. 件数に対して指数対数的に増加し, 初期化時に 10,000 件の単語読み込みで 4 秒程度かかるためこの方法での限界と考えられ, これ以上増える場合はデータベースなどを活用することを検討する必要がある. 辞書引き検査時間は単語数 5,000 件以上において 1 秒弱で変化がないことがわかる.増大しないため件数による制約は少ない.

7.2 正規表現パターン数の影響

正規表現パターンの処理は, 現状では順次処理を行なっているため, パターン数が影響を及ぼすと考えられる. 実験において, パターンが違うことで処理時間が増大するとは考えられないため, 同じパターンを複数回登録することで測定した. 測定結果を表 3 および図 3 に示す.

表 3　パターン数と解析時間

パターン数	解析時間(ms)
1	13
1000	150
10000	599
50000	3525
100000	6198

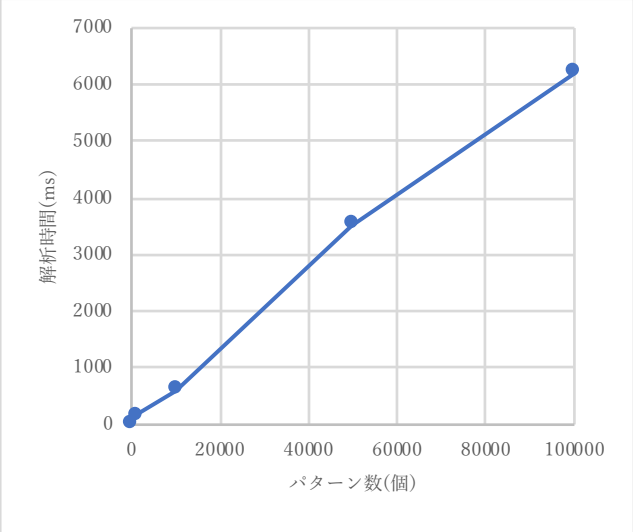

図 3　パターン数と解析時間グラフ

正規表現パターンと解析時間には直線的な関係があることがわかる. 10 万パターンに対して 6 秒程度で解析が終わる. 正規表現処理の時間は対話システムの応答速度に直接影響するため, 早ければ早いほどよい. パターン数に対して処理時間が比例的に増えることがわかるが, 次のような特徴があるため, パターン数の増加に対して考慮する必要は少ないと考えられる.

1. 正規表現パターンは並列処理が可能なため, スレッドで分散並列処理により短縮化が可能である.
2. 正規表現パターン処理は単語種類が一致したときのみ実施されるため, 抽出された意味属性種類の組み合わせに対して正規表現パターンが爆発的に増えなければ問題になりにくい.
3. 正規表現パターンは人手で作ることが基本なため, 数万種類にもおよぶ可能性はほぼありえない.

7.3 性能に関するまとめ

今回の施策において, パフォーマンス試験を通して以下のようなことが言える.

1. HashMap を基盤としたインメモリー・オントロージ辞書は, 起動時の読み込み設定時間が単語数に対して指数関数的に増加し, 辞書単語 10,000 個に対して 4 秒程度の初期設定時間がかかり, ロボットの起動時間の限界と考える.
2. 辞書の単語検索時間は, 4000 件未満では比例的に増加し, それ以上では辞書単語数に関係なく増加せず, 1 秒弱の検索時間がかかる.
3. 正規表現処理パターン処理時間は, 件数に従い比例的に増加するが, 比較的短時間で終わり, 1,000 パターンに対して 150ms 程度である.

8. おわりに

本研究では小型ロボットでの音声対話システムのエッジ側搭載を目的とした場合のフットプリントが大きすぎるという課題に対して, 実装サイズを小さくすることを目標に, OSS を多用する従来実装方法を参考にして, Java にて正規表現を活用した試作を行なった. 結果, 従来実装方法では 370MB だったフットプリントが 27KB まで縮小できた. 辞書単語 4,000 個, 正規表現 1,000 パターン程度のデータに対して, 意味属性抽出と文型マッチング処理を約 1 秒の実用的な速度で実現できることを示した.

今後, 処理速度を従来実装方法と比較する予定である. 解析精度の比較も必要と考えるが, 方式の検討が必要である. また, 意味属性抽出をデータベース化する, パターンマッチング処理をマルチスレッドにより並列化するなどして, 大規模辞書やパターン数増加に対してどのように効果的か検証することを検討している.

参考文献

[1] 竹元義美, 福島俊一, 山田洋志. 辞書およびパターンマッチルールの増強と品質強化に基づく日本語固有表現抽出, 情報処理学会論文誌, 2001.
[2] 須藤 克仁, 中野 幹生. “音声対話システムのための 統計的言語理解モデルの構成とその学習”, 言語処理学会年次大会, 2004. http://www.anlp.jp/proceedings/annual_meeting/2004/pdf_dir/C1-3.pdf
[3] 勝丸真樹, 中野幹生, 駒谷和範, 船越孝太郎, 辻野広司, 尾形哲也, & 奥乃博. 複数の言語モデルと言語理解モデルによる音声理解の高精度化, *電子情報通信学会論文誌 D*, *93*(6), 879-888. 2010
[4] 福林雄一朗, 駒谷和範, 中野幹生, 船越孝太郎, 辻野広司, 尾形哲也, & 奥乃博. 音声対話システムにおけるラピッドプロトタイピングを指向した WFST に基づく言語理解, 2008 *情報処理学会論文誌*, *49*(8), 2762-2772.
[5] Apache Lucene-GoSen. https://code.google.com/archive/p/lucene-gosen/ .
[6] MeCab. http://taku910.github.io/mecab/ .
[7] Apache OpenNLP. https://opennlp.apache.org/ .
[8] OpenFST. http://www.openfst.org/ .