【様式4】

テーマ名：単語の共起情報を利用したかな漢字変換システム
申請者名：石田岳志

【提案テーマ詳細説明】

1 なにをつくるか

既存の手法より高精度なかな漢字変換エンジンを開発する。

かな漢字変換を行う方法として現在主流なのは、単語ユニグラムと品詞バイグラムを用いてラティスを構築し、コストが最大または最小になるような経路を探索することで変換結果を得るというものである。しかし既存手法の欠点として、変換したいかな文字列以外の情報を変換に利用できないことと、2単語以上離れた位置にある単語の情報を利用できないことが挙げられる。

私は、変換候補に含まれる2単語の共起確率を利用することにより変換精度を向上させる手法を提案し、開発を行う。

1.1 かな漢字変換とは

かな漢字変換エンジンとは、入力されたかな文をかな漢字交じり文に変換するシステムであり、日本語入力システムや音声入力システムなどに搭載されている。

かな漢字変換を搭載した日本語入力システムには次のようなものがある。

- Google日本語入力 (オープンソース版はMozcとして公開されている)
- ATOK
- Microsoft IME
- SKK

1.2 かな漢字変換の手法

ここでは、「けんきゅうする」というかな文字列の変換を例として、かな漢字変換のアルゴリズムの解説を行う。

変換に用いられるアルゴリズムは複数の部分に分割できる。本章では各段階ごとの説明を行う。

1. 与えられたかな文字列を適当に分割する
2. 分割されたかな文字列に対応する変換候補(漢字やカタカナ)を列挙する
3. 変換候補となる単語を用いてグラフ(ラティスと呼ばれる)を構築する。変換候補の単語がラティスのノードに相当する
4. ラティスのノードとエッジにコストを与える
5. ラティスの開始点と終了点の間を結ぶ経路のうち、コストが最大または最小となるものを見つけ出す。経路上にある文字列をつなげたものが変換結果として出力される

1.2.1 与えられたかな文字列を適当に分割する

「このひとことでげんきになった」というかな文字列が与えられたとき、これを先頭から区切っていく(リスト1)。

```
/け/んきゅうする/
/けん/きゅうする/
…
```

```
/けんきゅう/する/
...
/けん/きゅう/する/
...
```
リスト1 入力されたかな文字列を区切る

1.2.2 分割されたかな文字列の変換候補を列挙する

分割された文字列はそれぞれ漢字やカタカナに変換することができる。

```
/家/んきゅうする/
/剣/窮する/
…
/研究/する/
...
```
リスト2 分割された文字列の変換候補を与える

1.2.3 変換候補となる単語を用いてグラフを構築する

経路を前方向にたどっていくとひらがな列の変換候補が現れるように、各単語をノードとしたグラフを構築する(図1)。このグラフはラティスと呼ばれる。

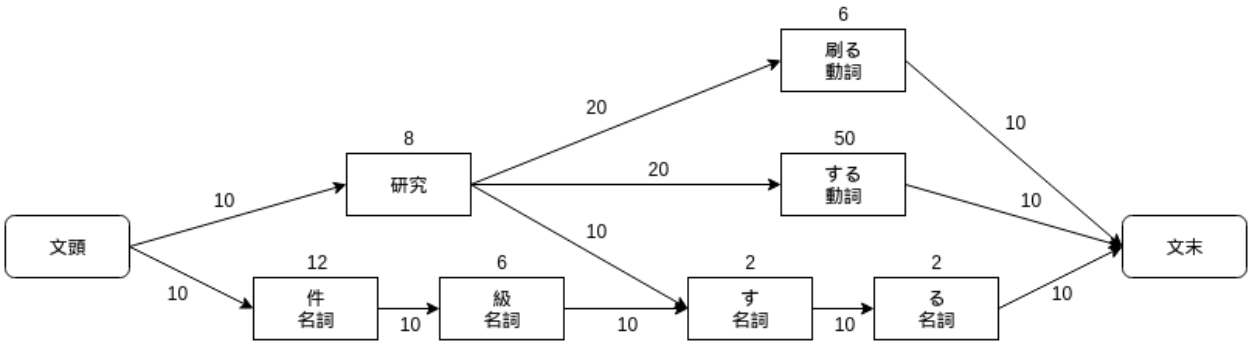

図1 ラティスの例

1.2.4 ラティスのノードとエッジにコストを与える

図1のように、各ノードとそれらを結ぶエッジにコストを与える。ここではコストが最大になる経路を変換結果として出力する場合を考える。

一般的な文章によく出現する単語は、変換結果の文章中に出現する確率も高い。よって一般的な文章の中に出現しやすい単語には高いコストを、出現することがあまり無い単語には低いコストを与える。

エッジにもコストを与える。単語Aから単語Bへのつながりがよく現れる場合には、AとBを結ぶエッジに高いコストを与える。例えば、「私」のあとに「の」が出現する確率のほうが、「私」のあとに「野」が出現する確率よりも高いため、「私」と「の」を結ぶエッジのコストは「私」と「野」を結ぶエッジのコストよりも高く設定される。

1.2.5 コストが最大になる経路を見つけ出す

ラティスの開始点(文頭)と終了点(文末)を結ぶ経路を探索する。経路上にあるノードとエッジのコストの合計が最大または最小になるような経路を見つけ出す。

1.2.4で頻出する単語と、つながりやすい単語間に高いコストを与えたため、ここでは経路のコストが最大になるような経路を探索する。コストが最大となる経路が変換結果として出力される。

探索にはビタビアルゴリズムなど、ダイクストラ法やそれに類似した手法が用いられる。

1.3 既存手法の問題点とその解決方法

既存手法には次のような問題点がある。

1. エッジのコストは実際には単語同士ではなく品詞同士のつながりで与えられる
2. 2単語以上離れた位置にある別の単語の情報を活用できていない
3. 変換する際には変換対象のかな文字列しか見ていない

それぞれ詳しく説明し、これらによって誤変換が発生すること、またこれらの問題がメモリ消費量を抑えるための設計に起因することを示す。さらに、上記3つの問題のうち、1は解決が難しいが、2と3に関しては、ユーザーの過去の入力や変換候補に含まれる単語の共起情報を利用することで解決できることを述べる。

未踏事業におけるプロジェクトでは、問題2と問題3を単語の共起情報を用いて解決する。

1.3.1 エッジのコストは品詞同士のつながりで与えられる

1.2.4において、「単語Aから単語Bへのつながりがよく現れる場合には、AとBを結ぶエッジに高いコストを与える」と述べたが、実際には「単語Aの品詞から単語Bの品詞へのつながりがよく現れる場合に」AとBを結ぶエッジに高いコストを与えている。

例えば、「期間」は名詞、「の」は助詞である。「期間」と「の」を結ぶエッジには、名詞の次に助詞が出現する場合のコストが設定される。一方で、「機関」も同様に名詞であるため、「機関」と「の」を結ぶエッジにも同様に、名詞の次に助詞が出現する場合のコストが設定される。よって、「期間」と「の」を結ぶエッジのコストと、「機関」と「の」を結ぶエッジのコストは等しく設定される(図2)。

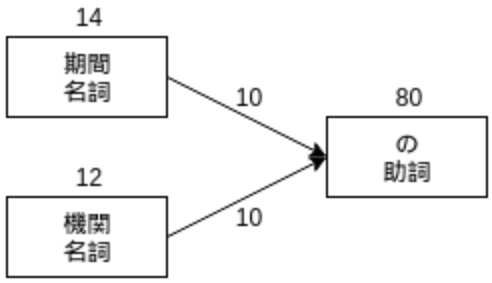

図2 品詞の出現パターンが同じなので、「期間」と「の」を結ぶエッジのコストと、「機関」と「の」を結ぶエッジのコストは等しい

既存手法では単語同士ではなく品詞同士のつながりを考慮しているため、各単語が持つ情報が失われてしまっている。仮に品詞同士ではなく単語同士のつながりを考慮することができれば、より詳細なコストの設定が可能になるはずである。

生じる誤変換の例

例えば、「たいさくをねる(対策を練る)」を変換すると、「対策を寝る」という誤った変換結果が得られてしまう。

「寝る」と「練る」はどちらも動詞である。よって、「を」と「寝る」を結ぶエッジのコストと、「を」と「練る」を結ぶエッジのコストは等しい。 もし「寝る」という単語に対して高いコストが設定されていたら、最大コストを与える経路は「寝る」を通ってしまう(図3)。

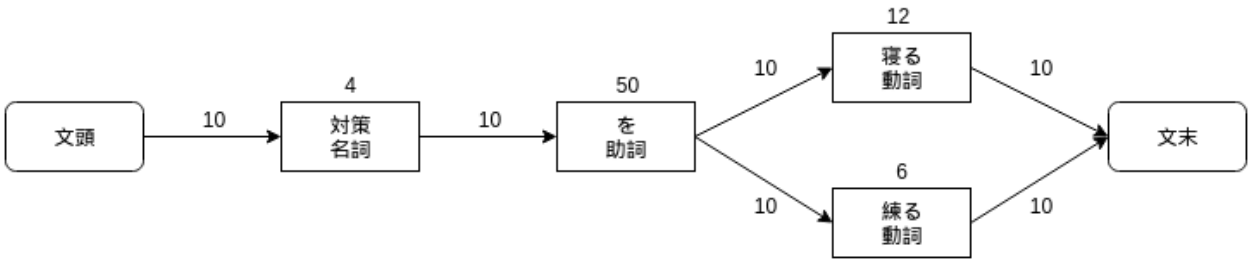

図3 「寝る」と「練る」はどちらも動詞であるため、「寝る」に高いコストが割り当てられると「対策を寝る」という変換結果が出力されてしまう

問題が発生する理由

単語同士ではなく品詞同士のつながりを考慮する理由は、メモリ消費量を抑えるため、そして、学習データの量を抑えるためである。

単語間のエッジのコストは学習によって求められる。学習データとなる文章中に、ある単語が出現したとき、その次にどの単語がどの程度頻繁に出現するかを記憶しておけば、エッジのコストが計算できる。例えば学習データ全体で「を」という単語が1000回出現し、そのうち「を」「する」という並びが60回、「を」「刷る」という並びが10回出現したとする。このとき「を」と「する」を結ぶエッジには60/1000=0.060、「を」と「刷る」を結ぶエッジには10/1000=0.010というコストを与えることができる。

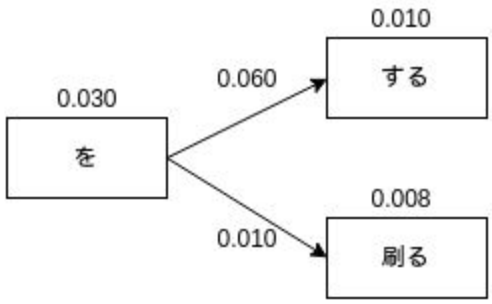

図5 単語同士の結びつきを考慮すると、
単語によってエッジのコストを変化させることができる

しかし、このように単語同士の結びつきを計算するには、全ての単語の並びのパターンの出現頻度を保存する必要がある。学習データに出現する単語の数は膨大であるため、単語の並びの出現回数を全て保存しようとするとメモリに収まりきらなくなってしまう。一方で品詞は細かく分類してもせいぜい数十種類程度である。学習データに出現する2品詞の並びのパターンは数百程度なので、2品詞の並びの出現回数ならばメモリに載せることができる。

単語同士のつながりを考慮しない理由はもうひとつある。例えば「を」「擦る」という単語の並びが学習データに出現しなかった場合、「を」と「擦る」を結ぶエッジのコストは0になってしまう。ユーザーが「をする」を「を擦る」と変換したかったとしても、エッジのコストが0であるため「を擦る」には変換されなくなってしまう。この場合に「を」と「擦る」を結ぶエッジのコストを与えるには、スムージングを行い、擬似的なコストを与えるか、「を」「擦る」という単語の並びが出現するまで学習データを増やすしかない。

1.3.2 2単語以上離れた位置にある別の単語の情報を活用できていない

上で述べたように、既存手法では2単語間の品詞のつながりは考慮されるが、3単語以上が連続する確率は考慮されていない。例えば、「私(名詞)」のあとに「の(助詞)」が出現する確率と、「の(助詞)」のあとに「家(名詞)」が出現する確率を考慮することはできるが、「私」「の」「家」の3単語が連続する確率や、それぞれの品詞が3つ連続する確率を考慮することはできていない。

生じる誤変換の例

先ほどと同様に、「みずをくむ(水を汲む)」を変換すると、「水を組む」という変換結果が出力されてしまう。

学習データ中に「水」「を」「汲む」という3単語の並びがある程度頻繁に出現しており、この3単語の並びの出現回数を記憶できていたとする。この場合、実際に変換を行う際には「水」が出現したことを考慮して、「汲む」のコストを上げることで、正確な変換を行うことができる(図4)。

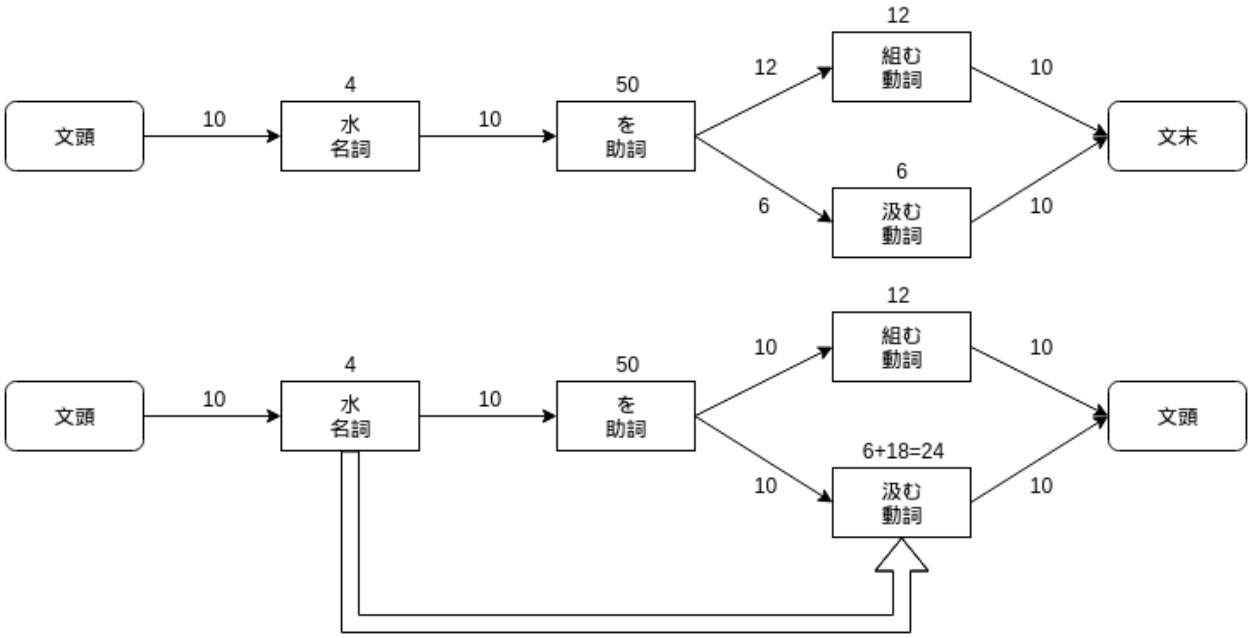

図4 上)2単語のつながりを考慮してコストを設定した場合
下)「水」という単語の情報を用いて「汲む」のコストを上げた場合

問題が発生する理由

これも1.3.1と同じく、メモリ消費量を削減するために生じる問題である。3品詞が連続するパターンを保存しようとするとメモリ消費量が増大してしまうため、3品詞以上のつながりは学習していない。

解決方法

1.3.3 変換する際に変換対象のかな文字列しか見ていない

「ほうそうしないんですか」というかな文字列が与えられたとき、「放送しないんですか」と「包装紙ないんですか」はどちらも変換結果として適切である。しかし、ユーザーがテレビ番組のレビューを書いている際には「包装紙ないんですか」は変換結果としてふさわしくない。

解決方法

ユーザーが「ほうそうしないんですか」と入力する前に「番組」という単語を入力していたら、「ほうそうしないんですか」は「放送しないんですか」に変換されることがふさわしいと考えられる。

過去数十単語程度のユーザーの入力を記憶しておき、それらの単語と変換候補に含まれる単語の共起確率を計算する。変換候補に含まれる単語のうち、過去に入力された単語と共起しやすいものに対して高いコストを割り当てれば、ふさわしい変換結果が得られるはずである(図5)。

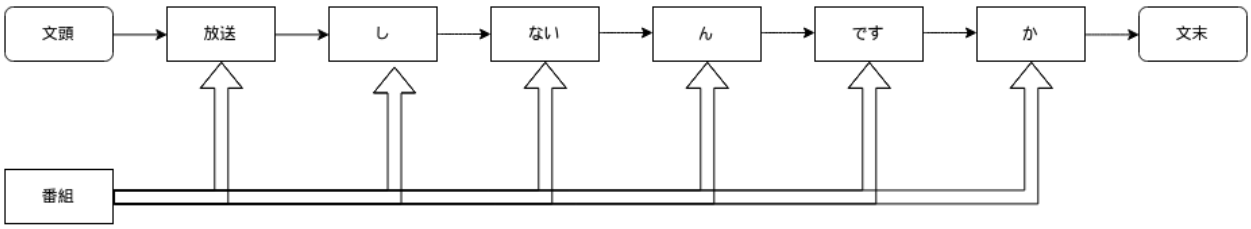

図5 ユーザーの過去の入力を利用してノードのコストを変化させる

1.4 共起情報を利用する方法

ここでは簡潔データ構造であるTrie木と、数値計算で用いられるSparse Matrixを用いて、メモリ消費量を抑えつつ単語の共起情報を保持する方法を解説する。

1.4.1 簡潔データ構造とは

簡潔データ構造とは、非常に少ないメモリ領域で情報を保持しつつ、情報に手軽にアクセスできるような機能を備えたデータ構造である。

かな漢字変換では、ひらがなを変換候補となる漢字やカタカナに変換するための辞書を保持している。メモリ消費量を抑えるため、この辞書は簡潔データ構造を利用して表現されている。かな漢字変換では簡潔データ構造としてTrie木がよく用いられる。

1.4.2 Trie木の性質

Trie木は文字列を保持することができる、木構造を利用したデータ構造である。

Trie木は図6のような形をしている。

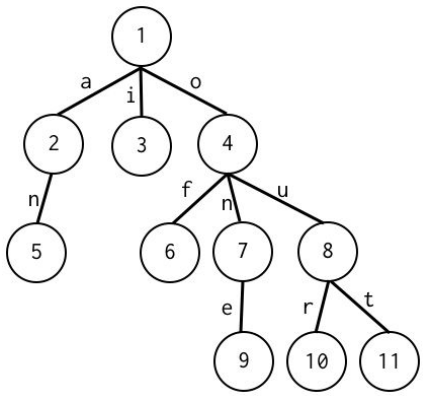

図6 Trie木 (真鍋宏史,2012)

木のエッジに文字が割り当てられることがTrie木の特徴である。この木には a, an, i, o, of, on, one, ou, our, out の10単語が格納されている。

Trie木が持つ性質を以下に列挙する。

- 別々の文字列の共通している部分を共通のエッジを用いて保持しているため、メモリ消費量を抑えることができる。
- 非常に高速にアクセスできる。木全体に格納されている文字数をN、検索したい文字の長さをMとすると、検索に要する時間のオーダーはO(M)である。つまり大量の文字を格納しても検索速度には影響しない。
- 検索したい文字列を与えると対応するノード番号が得られる。例えば図6の木に対してoneという文字列を与えると9という数字が得られる。また逆にノード番号を与えるとその番号に対応した文字列を取得することもできる。

実際にはTrie木は木構造をビット列で表現するためのLOUDSという手法と組み合わせて実装されるため、検索速度はO(M)とはならないが、それでもO(MlogN)程度に抑えることができる。

1.4.3 Sparse Matrixとは

Sparse Matrixとは、3本の配列を用いて行列を表現する方法である。端的に述べると、行列の非0の要素と、要素の行番号、要素の列番号をそれぞれ配列に格納することで行列をコンパクトに表現する手法である。要素の行番号と列番号も保持するため、行列の要素がほとんど0でないと、2次元配列を用いた行列の表現よりもメモリ消費量が増大してしまう。

詳しい解説はWikipedia(goo.gl/R5c75g)を見たほうがわかりやすいためここでは深く解説しない。

1.4.3 Trie木とSparse Matrixを用いて共起回数を保持する方法

Trie木を用いること単語をその単語に固有の番号(ノード番号)に変換することができる。よってTrie木を用いると、単語Aと単語Bの共起回数は、

- 単語Aに対応するノード番号
- 単語Bに対応するノード番号
- 単語Aと単語Bの共起回数

によって表現することができる。これらは、単語Aに対応するノード番号を行列の行番号、単語Bに対応するノード番号を行列の列番号、AとBの共起回数を要素とした行列として表現できる(図7)。この行列はほとんどの要素が0になるため、Sparse Matrixを用いるとコンパクトに表現することができる。

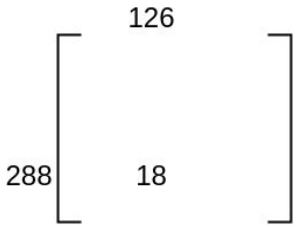

図7 例えば単語Aに対応するノード番号が288、単語Bに対応するノード番号が126、
AとBの共起回数が18のとき、行列の(288, 126)成分は18である。

1.5 未踏事業で実装するもの

未踏事業ではかな漢字変換システム全体をフルスクラッチで実装する予定だが、以下の部分については既に実装済みである。

- 2単語のつながりまたは2品詞のつながりを学習する部分
- 単語の出現頻度を計測する部分
- 経路探索アルゴリズム

[GitHub](https://github.com/IshitaTakeshi/KanaKanjiConversion/)

内部で用いるTrie木も実装し、ライブラリとして公開している。
[GitHub](https://github.com/IshitaTakeshi/DTrie)

Google日本語入力にはオープンソース版のMozcがあるが、C++で書かれているため、保守性が低くデバッグも難しい(コーディング規約が守られており、モジュール化もなされているためおそらく保守はしやすいが、C++で書かれているため原因不明のバグへの対処が難しい)。このためMozcに搭載されている機能をモジュールとして利用することはあるかもしれないが、Mozcに対するcontributionという形での開発は想定していない。

2 どんな出し方を考えているか

オープンソースソフトウェアとして公開する。デスクトップに限らず、モバイル端末でも動作するようにする。Google日本語入力と同等のプラットフォームで動作させることを目指す。

また、かな漢字変換に用いられているアルゴリズムは音声認識などにも応用できるため、有用な部分はできる限り切り出してライブラリ化する。

3 斬新さの主張、期待される効果など

既存のかな漢字変換エンジンよりも効率的な変換を実現できるようになる。

社会に目に見えるような変化を与えられるとは思わないが、あらゆる場所での作業効率を向上させることができる。また、かな漢字変換の内部で用いられているアルゴリズムは音声認識など他のタスクでも利用できるため、内部のアルゴリズムを切り出し、ライブラリとして公開することで技術に対する貢献もできると考えている。

4 具体的な進め方と予算

4.1 開発を行う場所

主に自宅や学校の寮、研究室で開発を行う。Laptopは常に持ち歩いているため、作業ができるなら場所や時間は問わない。

4.2 使用する計算機環境

自身のLaptopまたは研究室に置かれているデスクトップパソコンを用いる。研究室の環境は学校側からまだ伝えられていないため、ここで述べることはできない。自身のCPUのスペックは下に表記したとおりである。

CPU	Intel Core i5
メモリ	8GB
OS	GNU/Linux

Androidアプリとして動作させる場合には、自身が保有するAndroid端末を利用して開発を行う。

4.3 使用する言語とツール

言語は主にD言語を利用する。エディタはVimを用いる。

D言語を選択した理由は以下である。

- 読みやすく、テストも書きやすいため、保守しやすい
- 高速に動作する
- 実行環境をインストールする必要がないため、ユーザーに環境設定を要求せずに済む
- CやC++へのインターフェースが付随しているため、C/C++で書かれたライブラリを利用できる

4.4 開発線表

2016/6	2016/7	2016/8	2016/9	2016/10	2016/11	2016/12	2017/1	2017/2	2017/3
変換エンジン全体の実装を行う 最低限動作するものを完成させる				共起確率を用いて変換精度を高める機能を実装する 機能追加と並行して精度評価を行う					

	早く終わった場合には日本語入力としての諸機能を実装する (例えば「きょう」の変換候補として「2016/06/20」を挙げるなど) 入力履歴にはユーザーの個人情報が含まれる可能性があるため、 セキュリティに関連する機構もできる限り実装する

4.5 開発にかかわる時間帯と時間数

平日は午後4時以降、休日は基本的に終日作業可能である。毎週木曜日の午後はアルバイトに費やしているため作業はできない。長期休業中はインターンなどを行う可能性があるが、そうでない場合は基本的に終日作業可能である。

作業時間の目安として、平常授業がある場合は週10~20時間、長期休業中は週20~40時間、全体で500~800時間程度を見込んでいる。

卒業研究の内容はまだ決定していないが、かな漢字変換に取り組める場合は平日の昼も作業することが可能となる。

4.6 予算内訳

個人的な開発の延長であるため、有償のツールなどを使うことは今のところ考えていない。かな漢字変換においては学習用のコーパスが必要になるが、有償のものは数十万円、場合によっては100万円を超えるためそもそも買うことができない。このため学習データはWebから収集する。精度評価用のデータセットには変換前のかな文字列が必要になるが、Mecabや読み推定ツールのKAKASIを用いることでよみがなを得ることができるため恐らく問題ない。

5 提案者（たち）の腕前を証明できるもの

私は機械学習を趣味で学んでいる。機械学習は学術的に非常に面白く、また応用範囲も非常に広いことが魅力である。機械学習を学ぶ人が増えること、機械学習という分野そのものが発展すること、機械学習を応用したアプリケーションやサービスが開発され、利用されることを望んでいるため、初学者向けに記事を書いたり、ライブラリを開発して公開したり、勉強会に参加して発表を行ったりしている。

記事

Googleで「機械学習」と検索すると、私の書いた記事が検索結果の2位か3位に表示されるはずである。「機械学習をこれから始める人に押さえておいて欲しいこと」という記事がそれである。また他にも、初学者向けに簡単にかつ面白くテキスト分類を体験してもらえるような記事(ナイーブベイズでツンデレ判定してみた)や、Pythonプログラマが持つべき心構え”The Zen of Python”をサンプルコードを用いて解説した記事も書いている。

[Qiita](http://qiita.com/IshitaTakeshi)

オープンソース活動

機械学習のモデルや、かな漢字変換に用いられるTrie木を実装し、公開している。

高速かつ高精度なオンライン線形分類器であるSCW(Soft Confidence-Weighted Learning)をPythonとJuliaで実装し、ライブラリとして公開している。どちらもpipまたはJuliaに付属しているパッケージマネージャを用いてインストールすることができる。

かな漢字変換ではTrie木で辞書を保持する必要があるため、これも自分で実装しライブラリとして公開している(DTrie)。

機械学習ではデータをsvmlight formatと呼ばれる形式でファイルに保存することがよくあるが、この形式のファイルを読み込めるJuliaライブラリが無かったため自分で開発し公開した(SVMLightLoader.jl)。

[SCW](https://github.com/IshitaTakeshi/SCW)
[SoftConfidenceWeighted.jl](https://github.com/IshitaTakeshi/SoftConfidenceWeighted.jl)
[DTrie](https://github.com/IshitaTakeshi/DTrie)
[SVMLightLoader.jl](https://github.com/IshitaTakeshi/SVMLightLoader.jl)

勉強会

勉強会にもよく参加し、発表を行っている。

機械学習ハッカソンではSCWについて解説し、参加者が実装を行った(SCWが難しい方向けはパーセプトロンを実装した)。

comb meet up!ではThe Zen of Pythonを紹介した。

JuliaTokyoではSoftConfidenceWeighted.jlとそのパフォーマンスを披露した。

先月はKaggleのMalware Classification Challangeに関する勉強会に参加し、マルウェア分類における有用な特徴量の解説を行った。

[機械学習ハッカソン](http://mlhackathon.connpass.com/event/5607/)
[comb meet up!](http://connpass.com/event/7772/)
[JuliaTokyo](http://juliatokyo.connpass.com/event/21715/)
[Kaggle - Malware Classification Challenge勉強会](http://connpass.com/event/25007/)

ハッカソン

YahooのOpen Hack U 2014とJPHacks 2015に参加した。

Open Hack U 2014では、楽曲から特徴量を抽出し、音色の似ている曲を検索するシステム”Lyra”を開発した。

JPHacks 2015では予定と気温に応じて服を推薦するシステムを開発し、決勝に進出した。

[Open Hack U 2014](https://www.facebook.com/permalink.php?id=442958957595 16&story_fbid=803468996377869)
[JPHacks 2015 final](https://jphacks.com/informations_detail/jphacks-2015-final/)
[JPHacksでの成果物](http://devpost.com/software/tk28)

アルバイト

2014年4月から株式会社ウサギィで機械学習のアルバイトをしている。業務内容は与えられた課題を解決するためのアルゴリズムの検討、論文の調査、モデルの実装などである。

6 プロジェクト遂行にあたっての特記事項

特になし。

7 ソフトウェア作成以外の勉強、特技、生活、趣味など

VOCALOID関連の作品が好きである。あるクリエイターが作品を公開し、その作品のファンが作品に対して別の表現を与え、またその作品のファンが新しい表現を与え... という連鎖が起きており、素晴らしい文化だと思う。私もアプリ開発という手段でこの創作の環に入ったことがある。あとミクさんはとてもかわいい。

ソフトウェア開発以外の趣味は、卓球をすることと、モータースポーツを観戦することである。また最近は街を歩く際にその街の歴史を調べ、一人でブラタモリのようなことをしている。

8 将来のソフトウェア技術について思うこと・期すること

ソフトウェアの最大の利点は、他者に簡単に成果物や技術を譲渡できることである。一度成果物をGitHubなどで公開すれば、世界中の人々に対して貢献することができる。一方でこれは技術が簡単に流出してしまうという最大の弱点でもある。

サービスの広告料を収入とすることで内部技術を公開しても問題ない状態にしてしまうことがまず考えられるが、インフラ関連やミドルウェアを開発している企業が同様のことを行うのは難しい。インフラ関連の企業は保守によって収入を得ることが考えられるが、製品をプロプライエタリにして販売するのと同等の収入が得られるかというと疑問である。

私は現在、コピーレフトを利用することで、自社の技術を他社が利用しても、自社のみが不利にならないようにするという手法に関心を抱いている。

「ソフトウェアに関わる全ての人々が、収入を得て豊かな生活を送ることと、技術に対して貢献すること両立できる世界」を私は望んでいる。