令和 3 年度 卒業論文

置賜地域介護現場における
方言翻訳システムの研究

山形大学工学部

情報・エレクトロニクス学科

電気・電子通信コース

平成 29 年入学

17513258

二瓶友岳

指導教員

横山道央准教授

目次

第1章. 序論

1-1. 背景

昨今、日本の介護業界における人手不足は大きな課題であり、2025 年までに 37.7 万人の介護人材が不足すると言われている[1]。この課題に厚生労働省と経済産業省は、ロボット技術の介護利用における重点分野を策定し、「高齢者等とのコミュニケーションにロボット技術を用いた生活支援機器」もその中に含まれている[2]。また、地方の医療・介護現場では、患者の症状を詳しく伝えるために方言の使用機会が多く、方言による発言をしっかり受け止めることが安心感や信頼感に繋がる。人間関係・信頼関係の構築のために、方言によるコミュニケーションは、超高齢化社会において注目されている分野である[10]。

医療・介護現場の山形の方言に対して県外出身者や外国人はどのように方言を理解するかという調査では、県外出身者が 25.2%，外国人が 83.1%の割合で理解ができないと示されている[3] [4]。また、東日本大震災の際に被災地で活動した医療・福祉関係者へのアンケート調査では、約 80%が「方言がわかると有益だ」と回答していることからも、福祉現場での方言理解支援ツールが有用であるといえる。

インターネット上の調査では、難しい方言として山形方言が 4 位・5 位という結果がでており、多方言の機械翻訳に関する研究では、48 方言中 11 番目に翻訳難易度が高いとされている[7][8][9]。従って、人間も機械も理解することが難しい方言のうちの 1 つであると言える。これを機械翻訳により共通語へ変換し、介護現場で使われているコミュニケーションロボットの言語を認識する機能に対し、API などによってカスタマイズを行うことで、コミュニケーションロボットが方言を認識できる状態を目指すこととした。

これにより、県外や海外出身の介護従事者が介護施設利用者との信頼関係の構築をしやすくなることを期待する。執筆者は、山形県置賜地域においてコミュニケーションロボットを活用するプロジェクトに参加していることから、本研究では山形県の置賜地域で話される方言について機械翻訳のためのシステムについての研究を行うこととした。

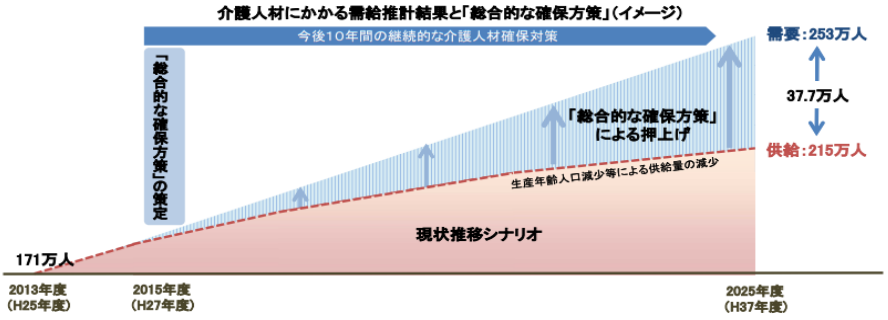

Fig.1 介護現場における 2025 年問題 （引用：[1]）

1-2. 全体像

本研究の全体像の理解を簡単にするために、前後関係を含めて本研究の位置付けについて説明する(Fig.2-1)。方言を機械処理にするためには、1. 方言の音声認識, 2.方言の形態素解析, 3.共通語への翻訳が必要であり、方言を共通語へ翻訳した後は、共通語と同様の手法で任意の自然言語処理を行うことができる。得られた結果をもとに、コミュニケーションロボットとの対話やコミュニケーションを使用するシステムとの連携ができるようにする。本研究では、Fig.2-1 にある「形態素解析」・「共通語に翻訳」に注目して研究を行った。

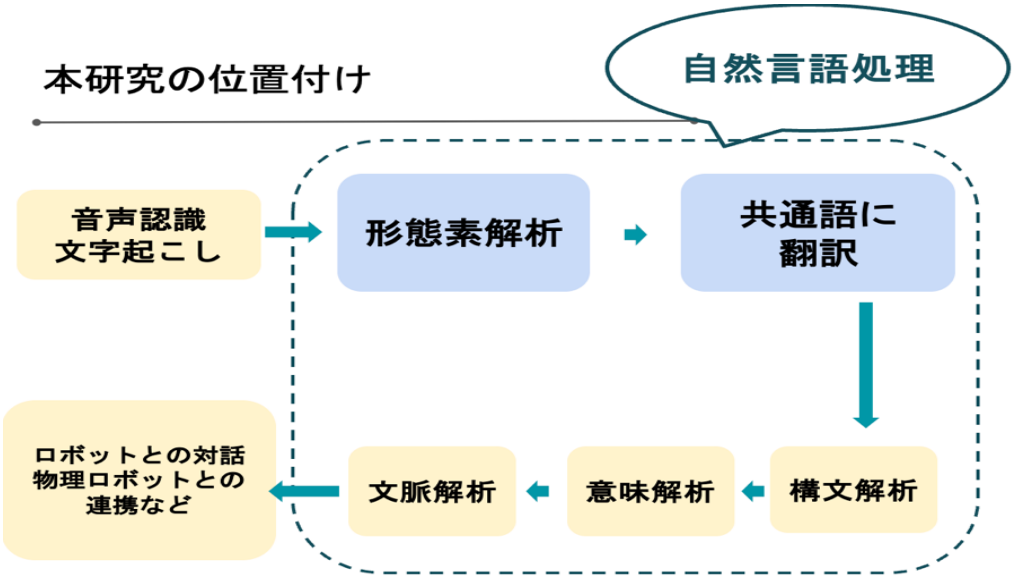

Fig.2-1 本研究の位置付け

「音声認識・文字起こし」では、置賜方言話者が方言を話す音声を正確に文字起こしする処理である。音声認識はオープンソースの音声認識ツールである Julius や Microsoft 社の Azure Custom Speech の使用によって行われるが、方言といった未知の単語の認識や雑音環境下における正確な文字起こしが課題となっている。

「形態素解析(Fig.2-2)」では、入力文を形態素に正しく分割することを行う。そのために、置賜方言の情報を辞書化する。その後、どの程度正しい形態素に分割できたかという評価を行う。

「共通語に翻訳(Fig.2-3)」では、置賜方言と共通語を置き換えることを基本の機能とする方言翻訳システムを開発する。置賜方言の情報とそれに対応する共通語の情報を辞書化することで、前述の形態素解析にて分割された形態素を共通語に置換する処理を実施する。また、どの程度正確に共通語に翻訳できたかという評価を行う。次章以降で詳細の理論と開発手法を説明する。

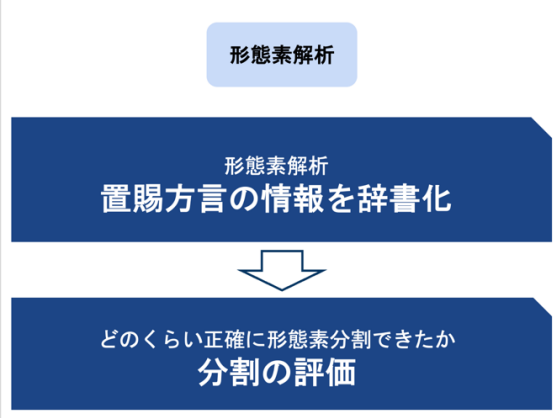

Fig.2-2 形態素解析

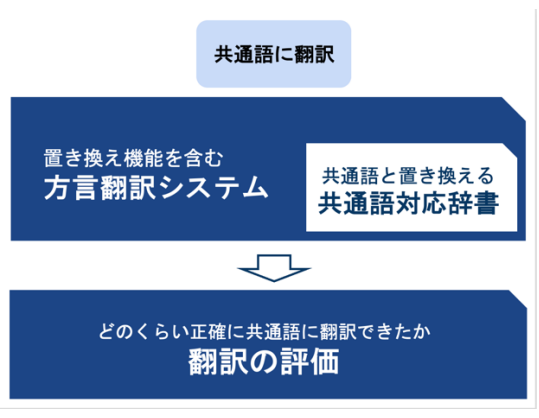

Fig.2-3 共通語に翻訳

1−3. 用語の解説

1-3-A. 自然言語処理

自然言語処理とは、人間が使用する自然言語を機械で処理・分析するための方法である。流れは、1. 形態素解析, 2 構文解析, 3 意味解析, 4 文脈解析の順で行われ、それらの結果は、大量のテキストデータの解析や非構造化データの処理として、機械翻訳・対話システム・音声の文字起こし・文章要約などに用いられる。

1-3-B. コーパス

コーパスとは、自然言語の例文を大量に集めたもの・音声データを集めたもの・文章を集めたものなど、自然言語処理のために大量に集めたデータのことを指す。また、注釈付きコーパスとは、コーパスに品詞タグや構文構造を付加したコーパスを指す。後述の「置賜 Dict」と「SudachiDict」は注釈（品詞タグ）付きコーパスに該当し、「実験用置賜方言例文コーパス」は、例文コーパスに該当する。

1-3-C. 形態素解析

形態素解析とは、日本語などの単語の区切り目の判別がつかない文章を形態素ごとに分割することを指し形態素解析器にて実施する。日本語の形態素解析器は、ChaSen, MeCab , Sudachi などがあり、様々な研究機関が協力してオープンソースソフトウェアとして公開しているものがほとんどである。一般に、形態素解析は、必要な情報を付与された辞書をもとに、最小コスト法にて実施されている。図(Fig.3)のように、形態素解析器ではまず形態素ごとに分割を行う形態素解析を実施し、それらに対して注釈付き辞書をもとに品詞の情報を付与する処理が行われる。この際に使用される辞書には、システム辞書とユーザー辞書があり、システム辞書は形態素解析器ごとに使用できるものが異なる。また

使用者が自由に語彙を追加できるものをユーザー辞書という。

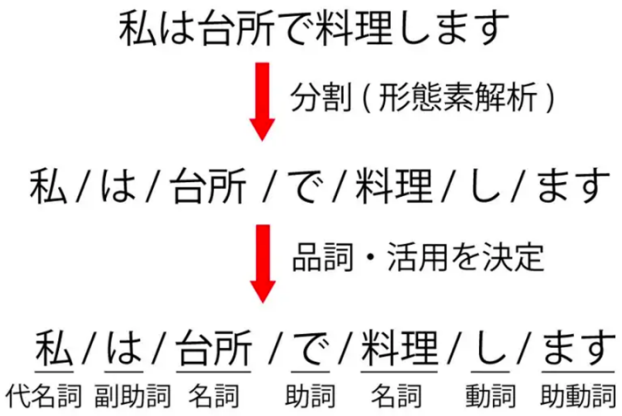

Fig.3 形態素解析の流れ（引用：[27]）

1-3-D. 最小コスト法

形態素解析を行う最小コスト法は以下のルールによって行われる方法である。

[1] 入力文を辞書にある単語ごとに区切る。

[2] すべての単語の区切り方の全パターンを抽出する。

[3] すべての単語に付与されている生起コストと連接コストの和をパターンごとに算出。

[4] 最も小さいコストのパターンを正解の分割とする。

例えば、以下の図(Fig.3)に示すように、「すももももも」という文を辞書に従って分割すると、「すもも / も / もも 」・「すもも / もも / も 」という 2 パターンが抽出される。このとき、あらかじめ「すもも」・「もも」・「も」にはそれぞれ連接コストと生起コスト情報が付与されており、これらのコストの総和が小さい「すもも / も / もも 」が正解として出力されるのである。以上のように、最小コスト法を使った形態素解析器は、形態素解析器に日本語の文法ルールなどの情報はなく、あらかじめ付与されたコストの最小値を算出することが行われる。

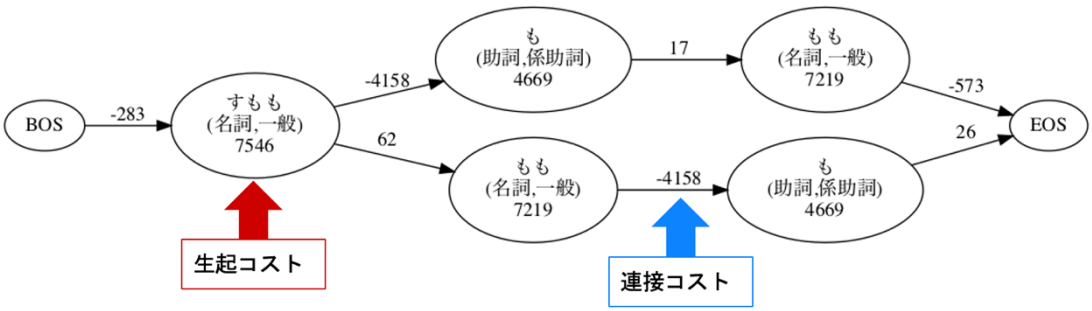

Fig.4 最小コスト法（引用： [26]）

1−4. 関連研究

1-4-A . 先行研究

山形県内の方言の機械翻訳に関する研究は、横山昌一教授らによって、研究が行われており、これまで主に庄内方言・村山方言・置賜方言についての基礎的研究がされ、村山方言については、共通語から方言・方言から共通語への双方向について研究されていた。方言から共通語に変換する研究では、形態素解析器 ChaSen を使用し、結合価と意味素性情報を付与して、辞書を作成し、共通語の単語と方言の形態素を単純置換する手法を使用していた。[32] – [38]

インターネット上では、置賜方言をまとめた情報の掲載が多くなり、2007 年には約 8000 語が収録された紙の辞書が出版され、置賜方言の保存に向けた地域住民の意識の高まりが起きている。これには、2004 年公開の映画「スウィングガールズ」に置賜方言が使用されていたことも後押ししていると考える。[10] – [18]

横山昌一教授らが主に山形県の方言の機械翻訳について研究を行っていた 1990 年代から 2000 年代前半では、置賜方言に関する情報が少なく、置賜方言のコーパスを作成することが、いまよりも難しかったことが予測できる。

1-4-B.　置賜方言

山形方言は、村山・置賜・庄内・最上の 4 地域に大きく分類され、それぞれ地域特有の方言が使われている。置賜方言は、南東北方言に属すると言われている。地理的に隣接する関東方言や南東北諸方言の影響も及んでいる。以下に簡単に文法の特徴を述べる。[11]

動詞は、サ変動詞についてラ行五段化の傾向がみられる。活用語の主な特徴は、主として後続の助動詞による。基本的には共通語と同じであるが、ネ(打消)、ラセル(使役)などで特徴がある。[11]

格助詞は、主格・対格が省略される特徴があり、方向・受け手・場所・目的には、(サ)が使われる。接続助詞には、(カラ・コンダラ・コンジャ・ケンドモ・タテ)が使われている。終助詞には、(ケロ・ケンニガ・ダズ・ゴデ) が挙げられる。[11]

共通語で使われている語彙の一部が濁音化・促音便化・省略されることが頻繁に起きる。

1-4-C.　形態素解析器 Sudachi [19] [20] [21] [22]

本研究で使用する形態素解析システム Sudachi は、2017 年に公開された新しい日本語形態素解析システムである。株式会社ワークスアプリケーションズのワークス徳島人工知能 NLP 研究所が開発した商用利用可能なオープンソースの形態素解析器であり、今後 10 年にわたり継続的に開発、更新していくことをプロジェクトの目標としている。主な特徴を遂に紹介する。

[正規化機能]

送り仮名・字種・異体字・誤用といった表記の違いを正規化する機能がある。これまでの形態素解析器よりも実用に使いやすいといえる機能である。例えば、「ふいんき」・「雰囲気」・「ふんいき」は、正規化機能を使うと「雰囲気」と出力され、「シュミレーション」・「シミュレーション」は、「シミュレーション」と出力される。

[分割モード]

Sudachi には、A(短単位)/B(中単位)/C(長単位) [28] の分割モードが提供されており、分割の細かさを変更することによって様々な場面で活用することができる。これらの組み合わせによって様々な用途に対応でき、検索用途は A と C、マイニングは C、語彙調査は B [29]を使うことで最適な出力が得られる。「選挙管理委員会」をそれぞれのモードで分割すると、A：「選挙/管理/委員/会」,B：「選挙/管理/委員会」,C：「選挙管理委員会」となる。

[Sudachi Dict]

Sudachi に標準搭載されているシステム辞書として同社が開発している大規模辞書の SudachiDict である。従来の形態素解析辞書では、継続的な更新や新語への対応が不十分などの理由から実用で使うことが難しかったが、SudachiDict は、形態素解析器 Sudachi のための辞書であり、汎用的に使える大規模かつ高品質な言語資源を目指して開発されている。国立国語研究所による超大規模なコーパス「NWJC」を利用して学習した 258 億語規模のコーパスにて学習を実施し、数ヶ月に 1 度専門家により継続的に更新している特徴がある。規模の大きさによって、small・core・full と 3 つの辞書が提供されているが、本研究では最も語彙数が多い full を使用する。

[ユーザー辞書]

Sudachi には、ユーザーが任意の未知語や優先的に形態素分割を行いたい語彙を追加できるユーザー辞書という機能がある。未知語とは、システム辞書に追加されていない語彙であり、この場合は、Sudachi Dcit に追加されていない語彙を未知語という。

注釈の付与ルールは、unidic-mecab 2.1.2 と Sudachi ユーザー辞書作成方法[20]に準拠して作成する必要がある。1 つの単語あたり{0 見出し, 1 左連接 ID, 2 右連接 ID, 3 コスト, 4 見出し（解析結果表示用), 5 品詞 1, 6 品詞 2, 7 品詞 3, 8 品詞 4, 9 品詞（活用型), 10 品詞（活用形), 11 読み, 12 正規化表記, 13 辞書形 ID, 14 分割タイプ, 15 A 単位分割情報, 16 B 単位分割情報}の 17 情報を付与する。

[同義語辞書]

同音意義語辞書が提供されており、Sudachi/Sudachipy の出力に同義語展開をするための

ライブラリとして Chikkar/Chikkarpy が提供されている。例えば、「閉店」の場合、「クローズ」・「close」「店仕舞い」が出力される。

1-5. 本研究の狙い

本研究では、置賜方言を機械翻訳するシステムを開発し、性能の向上を目指す。機械翻訳に必要なコーパスの開発を併せて行い、評価を行う。

第2章. 理論

一般的に、自然言語処理を行う際、入力文を形態素解析器のシステム辞書に基づき、形態素解析を行うことによって形態素ごとにわかち書きすることが必要だが、方言の機械翻訳を行う際、システム辞書に方言は含まれておらず、正確に形態素分割が出来なくなることがある。そのため、本研究では置賜方言の情報を収集し、形態素解析器のユーザー辞書として使用可能にするための注釈を付与し、ユーザー辞書と従来のシステム辞書に基づき、正確な形態素分割を行う。形態素分割の結果、置賜方言として分割された形態素を対応する同じ意味の共通語と置き換えることで方言翻訳を実施することとし、これを置賜方言翻訳システムの基礎とする。その後、置賜方言の特徴に基づき、単に置き換えるだけでは正確な翻訳ができない部分の翻訳性能の向上を目指す。(参考図 Fig.5)

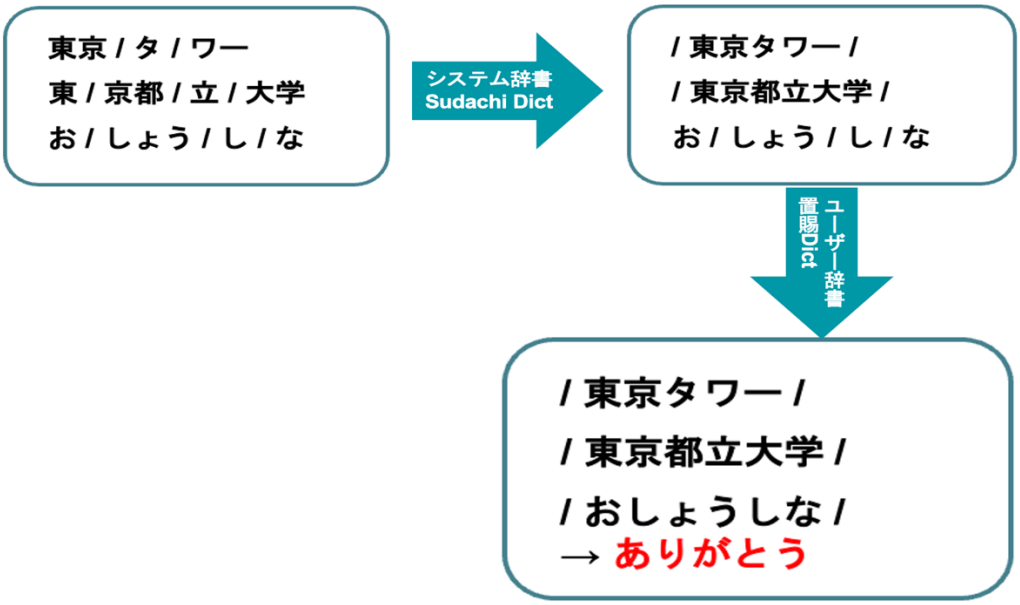

Fig.5

第3章. 開発

方言翻訳システムの構築と評価を行うため、置賜方言を認識した形態素分割をできるようにするための注釈付き置賜方言コーパスの開発を行い、どれほど性能が向上したかの評価システムを開発する。また、方言翻訳システムを開発し、それはどの程度正確に翻訳できるかという評価システムを開発する。評価するための入力文と方言翻訳の正解とする文を収集したコーパスの開発も行う。また、置賜方言と共通語の置き換えでは正確に翻訳できない特徴の一つとして、省略されている格助詞を補完するシステムを開発し、その場合の性能評価もおこなった。評価手法は、2 種類の機械翻訳評価手法と 1 種類の人間評価手法を採用した。以下に、作成した各システム・コーパスの説明を行い、評価手法の定義を行う。また、開発には、Pyhton3 系を使用し、Sudachi の Python 版である Sudachipy を使用して行った。出力は、csv ファイルまたは、matplotlib によって作図されたグラフである。

3-1. 注釈付き置賜方言コーパス（ 置賜 Dict ）

置賜地域で現在も使用されている方言と介護現場で使われている方言をインターネット上の個人ブログやサイト [11] - [18]を中心に収集を行い、収集した置賜方言の 1199 語に品詞情報を含む注釈付きコーパス(以下、置賜 Dict)を作成した。作成方法は、1-4-C にある[ユーザー辞書]に基づく。17 情報の項目のうち、5{品詞}から 10{品詞(活用形)}の項目は、任意の文字情報を付与することが可能なため、置賜 Dict では、10{品詞(活用形)}の項目を方言情報とし、「方言」という情報を付与することで、形態素ごとに共通語に置換する際、方言か判断させることとした。また、12{正規化表記}の項目に 0{見出し}に対応する共通語を記入し、置換する文字列を指定することとした。以下に、簡易版の置賜 Dict を示す。完全版の置賜 Dict は、[付録資料 1]に示す。

見出し： 0{見出し}	連接 ID 1{右連接 ID}， 2{左連接 ID}	品詞： 5{品詞}	方言情報： 10{品詞(活用形)}	共通語： 12{正規化表記}
おしょうしな	5687	感動詞	方言	ありがとう
やめる	5160	形容詞	方言	痛い
あぐど	4785	名詞	方言	かかと

Fig.6 簡易版：置賜 Dict

3-2. 置賜方言翻訳システム

置賜方言から共通語への翻訳処理は、次の流れで行う。

[step1] 形態素解析

テキストにて入力された文書を置賜 Dict と Sudachi Dict の情報を元に形態素解析器 Sudachi にて、形態素ごとの分割を行う。

[step 2] 語彙変換

形態素が方言の場合、辞書引きにより語彙変換を行う。3-1 注釈付き置賜方言コーパス(置賜 Dict)にて、【10{品詞(活用形)}の項目を方言情報とし、「方言」という情報を付与する】を行なったことにより、方言の判定を行うことができる。「方言」という情報が付与されていた形態素の場合は、3-1 の【12{正規化表記}の項目に 0{見出し}に対応する共通語を記入し、置換する文字列を指定することとした。】により、置賜 Dict を参照し、指定した共通語に置換する。

[step 3] 形態素結合

形態素を結合し、出力する。

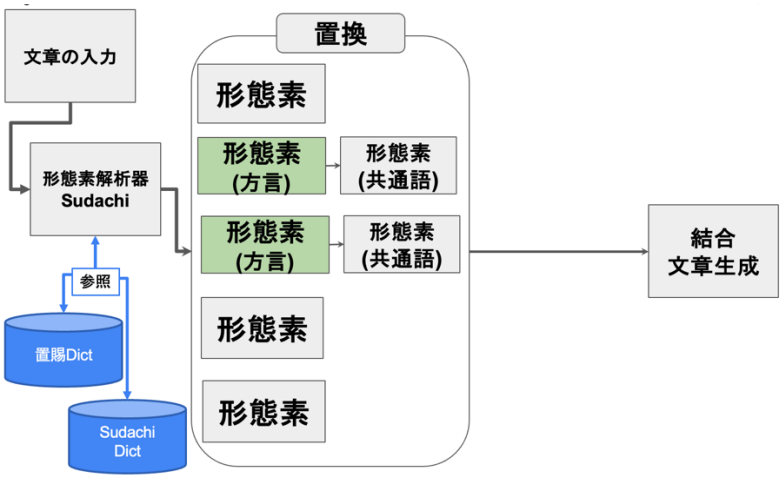

Fig.7-1 置賜方言システム

3-3. 格助詞補完システム

格助詞が頻繁に省略される特徴を持つ置賜弁の翻訳に対して、名詞・代名詞の次に動詞・形容詞が続いた場合は、それらの間に格助詞が省略されたとみなし、頻繁に省略される主格・対格・目的の格助詞（が・を・に）を補完する手法をとる。本研究では、Step2 の次の

処理として、試験的にランダムで補完し、どの程度性能が変化したか評価することでより今後の研究に活かしたいと考え実装した。(参考図：Fig.7-2, Fig.7-3)
このシステムの可能なカスタマイズとして、1. 補完する格助詞の種類を指定すること 2. 重み付きランダムにより指定した格助詞が出現する頻度を指定することができる。実験では、(が・を・に) の格助詞にそれぞれ均等な重み付けをしてランダムに補完した。

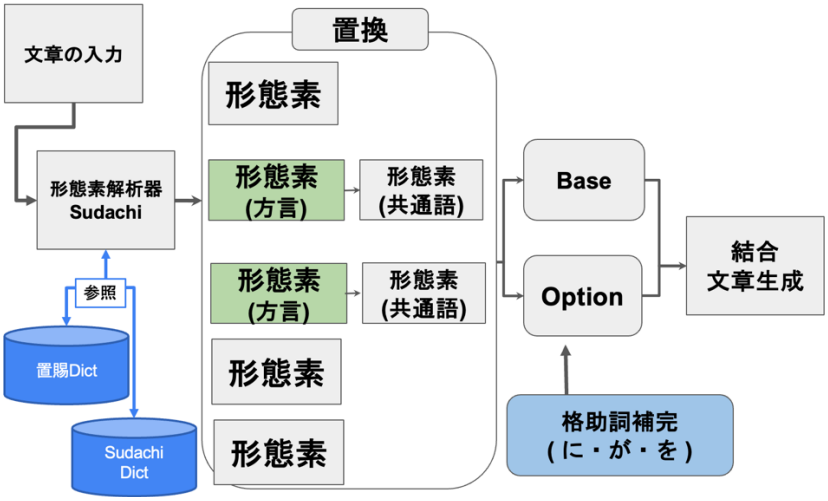

Fig.7-2　格助詞補完機能を追加した置賜方言翻訳システム

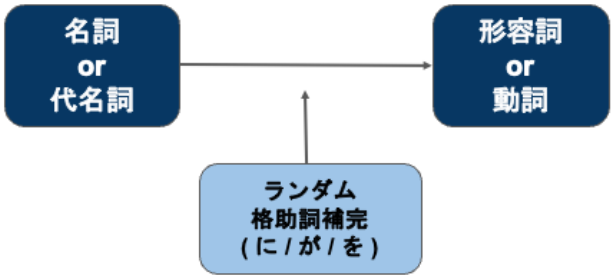

Fig.7-3 格助詞補完機能の詳細

3-4. 実験用置賜方言例文コーパス

置賜方言を翻訳するシステムを開発し、性能評価実験を行うにあたり実験用の入力文データが必要である。介護現場で使われている置賜方言を中心に 170 の例文を後藤らの研究を主に引用し、作成した。置賜方言が含まれている文章を人手にて「,(カンマ)」で形態素に分割し、それを形態素解析の正解データとした。また、人手にて共通語翻訳をした結果を置

賜方言の機械翻訳した際の正解データとした。[付録資料 2]

3-5. 分割性能自動評価システム

形態素の分割性能について評価するために、それぞれに人手にて正解とする分割情報を付与し、出力結果と比較することでどの程度正確に分割が行えているかを精度と再現率にて計算し、その 2 つの調和平均である F-score にて判断することとする。以下に、それぞれの計算式の定義を示すが、分割結果が過剰に分割されていると精度が小さくなり、分割不足があると再現率が低い数値となるため、それらの調和平均である F-score を算出することで単語抽出の統一された評価基準として用いる [21]。システムは、Sudachi の分割モード A によって形態素分割し、得られた形態素数を「システムが出力した形態素数」、正解とする形態素分割との一致数を「正しく抽出された形態素数」、正解とする分割の形態素数を「正解の形態素数」として算出した。

$$精度 = \frac{正しく抽出された形態素数}{システムが出力した形態素数} \text{-(6.1)}$$

$$再現率 = \frac{正しく抽出された形態素数}{正解の形態素数} \text{-(6.2)}$$

$$Fscore = \frac{2 * 精度 * 再現率}{再現率 + 精度} \text{ – (6.3)}$$

3-6. 翻訳性能の自動評価

3-6-1. F-score による自動評価

システムから出力された翻訳後の文章（以下、参照文）と正解とする共通語の文章（以下、正解文）がどの程度一致しているか評価をする。これは方言の機械翻訳を行う研究において多くの場合使用されている評価手法である。それぞれの文に対し、Sudachi Dict のみを元に再度、形態素分割し、品詞情報を付与し、単語と品詞情報が一致した単語の数によって、正しく抽出されたと判断することとした。以下に定義の計算式を示す。前述した[3-5. 分割性能自動評価システム]と同様の理由から F-score を使用する [21]。

一致した形態素の数を「システムが出力した形態素と品詞情報の数」、正解とする形態素

と品詞情報の数を「正しく抽出された形態素と品詞情報の数」、正解とする形態素と品詞情報の数を「正解の形態素と品詞情報の数」として算出した。

$$精度 = \frac{\text{正しく抽出された形態素と品詞情報の数}}{\text{システムが出力した形態素と品詞情報の数}} \quad \text{-(6.4)}$$

$$再現率 = \frac{\text{正しく抽出された形態素と品詞情報の数}}{\text{正解の形態素と品詞情報の数}} \quad \text{-(6.5)}$$

$$\text{Fscore} = \frac{2 * 精度 * 再現率}{再現率 + 精度} \quad -(6.3)$$

3-6-2.BLEU による自動評価

[BLEU]

BLEU とは、[24]にて紹介された手法である。システムから出力された翻訳後の文章（以下、翻訳文）と正解とする共通語の文章（以下、参照文）がどの程度一致しているか文脈を含めて評価できる。これは、ニューラルネットやディープラーニングなどを使用した自然言語処理研究の際に多く用いられる評価手法である。n-gram によって正解文章と機械翻訳が生成した文章を比較し、数値化する手法である。翻訳前と翻訳後の言語の文法が近い言語の翻訳評価に有効とされている。昨今、BLEU にはより正確に自動評価を行う手法が研究されており、7 つの平滑化手法がある。今回は、 [25]で紹介された最新かつ最も人間の評価に近いとされている 7 つ目の手法を使用して平滑化を行い、自動評価を行った。

[n-gram]

n-gram とは、n 個の連続する単語や文字のまとまりを表すもので、文脈を表す素性として使われるものである。1-gram(unigram)は、任意の単語の直前の 1 単語、2-gram(bigram)は、任意の単語の直前の 2 単語といったように、参照文と翻訳文の任意の単語について、直前の n 単語を含めて比較することで、同じ単語で意味が違う場合はネガティヴな判定をし、前後の単語関係も一致しているとポジティブな判定を行うことができ、その文章の文脈を含めて同じ意味の文章かどうか判定することができる考え方である。

[Natural Language Toolkit]

Natural Language Toolkit（ 通称: NLTK)は、自然言語処理を行う Python のライブラリ[31]であり、BLEU による自動評価を行える機能(BLEU-score)が提供されている。任意の n-gram を指定することができ、1 つの翻訳文に対し、任意の数の参照文を指定することができる。また、任意の平滑化手法を指定することができる機能である。

3-7. 人手による翻訳性能の評価

人手評価においては、文法性・共通語らしさ・意味の普遍さを人手によって判断するもので、日本語ネイティブ話者 1 名の評価者によって評価することとした。文法性・共通語らしさは、1~3 点(3 が満点)、意味の不変さは変化している(1)・変化していない(2)の 2 択で評価を行う。各々について平均値を算出する。また、どれだけ正しい翻訳が行えたかを調べるためすべての指標が満点の割合も併せて算出する。 [23]

第4章. 実験

方言翻訳を行うにあたり、以下の 2 つの実験を行うこととした。
1. 分割性能評価実験
2. 翻訳性能評価と格助詞補完による翻訳性能の評価実験

4-1. 分割性能評価実験

4-1-A. 実験目的

方言を正確に翻訳するためには、文章を正確に形態素分割する必要がある。そのためには、Sudachi で形態素分割する際、置賜 Dict を使い、参照することで可能となる。本実験では、置賜 Dict の有無による形態素解析器の形態素分割性能の変化について注目する。

4-1-B. 手順

[手順 1]

形態素解析器 Sudachi に標準で備わっているシステム辞書 Sudachi Dict のみを形態素解析時に参照し、[3-4. 実験用置賜方言例文コーパス]にある 170 の文章を[3-5. 分割性能自動評価システム]に入力し、出力結果を得る。

[手順 2]

形態素解析器 Sudachi のユーザー辞書として、[3-1. 置賜 Dict]を準備する。[3-4. 実験用置賜方言例文コーパス]にある 170 の文章を[3-5. 分割性能自動評価システム]に入力し、Sudachi Dcit と置賜 Dict を参照して、形態素解析し、出力結果を得る。

[手順 3]

[手順 1],[手順 2]からそれぞれ 170 の結果が出力される F 値の分布をヒストグラムに表し、中央値・平均値を算出し、比較する。それぞれの結果を結果①,結果②とする。

4-1-C. 結果

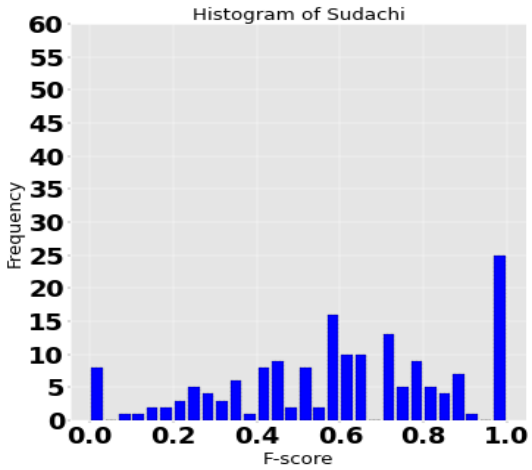

Fig.8-1 結果①の F 値の分布

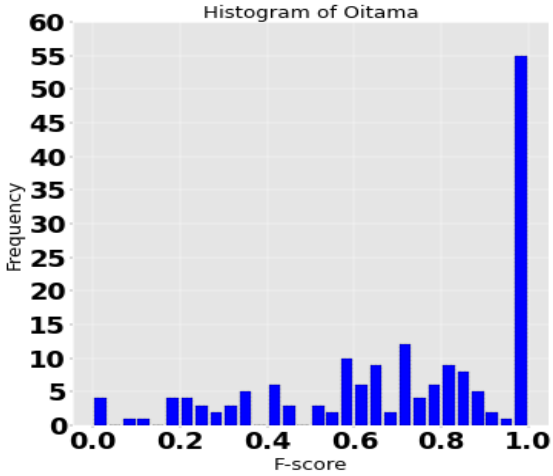

Fig.8-2 結果②の F 値の分布

	結果①	結果②	改善率
平均値	0.60	0.71	18.3%
中央値	0.60	0.77	28.3%

Fig.8-3 結果①・結果②の平均値・中央値と改善率

結果より、F-score=0.967~1.0 となる分割が行われた文が結果①から結果②で、2.2 倍となっており、全体として平均値 18.3%，中央値 28.3%の改善が行われたことがわかる。

4-1-D. 考察

Sudachi Dict のみを参照して行なった結果①と Sudachi Dict と置賜 Dict を参照して行なった結果②を比較すると、全体的に F-score=1.0 に近づいた文章の数が増え、全体として改善傾向が見られた。従って、置賜 Dict の追加によって分割性能が良くなったといえる。一方、結果①と結果②を比較し、F 値が向上していない文も一定数あった。

F 値が向上しなかった文の原因は、2 つに分けられ、1. 置賜 Dict に追加されていない単語が文に含まれていた場合・2. 置賜 Dict に追加されているが反映されていない場合であった。前者のケースは、その単語を新たに追加することによって正確に分割される文章数が向上すると考える。また、後者のケースは、該当する単語の連接コスト・生起コストを低くして、最小コストとなるように、調整をすると正確な分割結果になると考える。

4-2. 翻訳性能評価と格助詞補完による翻訳性能の評価

4-2-A. 実験目的

SudachiDict と置賜 Dict によって形態素解析を行い、置賜 Dict にある形態素の場合は、共通語に置換するシステムを開発した。このシステムによる翻訳性能の評価を行う。また、置賜方言の特徴を踏まえ、格助詞が欠落する部分に格助詞を補完する機能を追加開発した。この追加機能による性能の変動を比較、評価をする。

4-2-B. 手順

[手順 1]

実験用置賜方言例文コーパスの 170 例文を置賜方言翻訳システム(以下、Base)を通して、置賜方言にそれぞれ翻訳を行う。

[手順 2]

実験用置賜方言例文コーパスの 170 例文を格助詞補完機能を追加した置賜方言翻訳システム(以下、Option+)を通して、置賜方言にそれぞれ翻訳を行う。

[手順 3]

[手順 1],[手順 2]から出力された結果を F-score, BLEU-score,人手評価の 3 つの評価手法にて算出し、比較する。それぞれの結果を結果③,結果④とする。

4-2-C. 結果と考察

[F-score による自動評価]

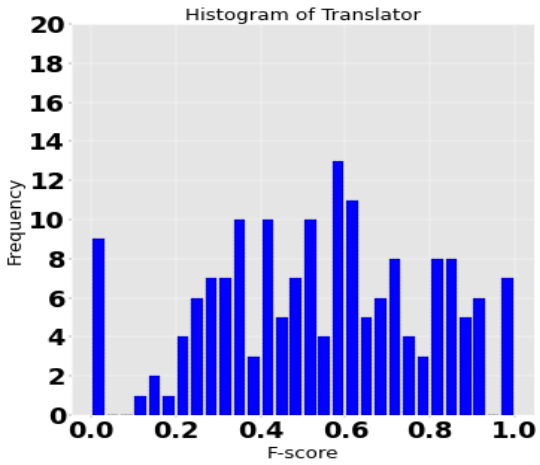

Fig.9-1　結果③の F-score の分布

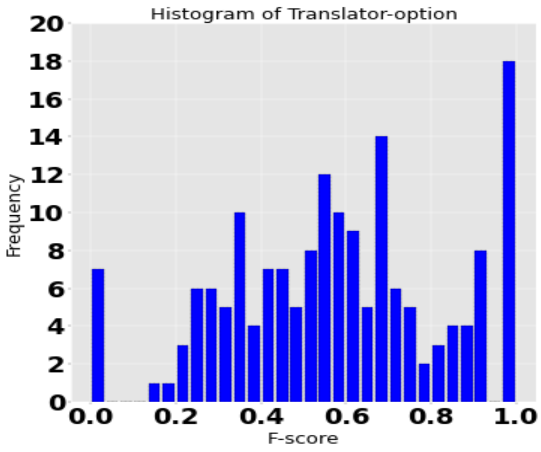

Fig.9-2　結果④の F-score の分布

	結果③（ Base ）	結果④（Option+）	改善率
平均値	0.54	0.58	7.40%
中央値	0.55	0.57	3.63%

Fig.9-3　結果③・結果④から得られた F-score の平均値・中央値と改善率

結果より、Base と Option+を比較すると、Option+により F-score＝0.967~1.0 となる文が結果③から結果④で、2.57 倍となり、全体的に F-score=1.0 に近づき、改善された文の割合が多い傾向にあった。一方、結果③における F-score＝0.0～0.033 の文については、結果④でも同様に F-score が低く、結果③から結果④で、F-score=1.0 に近づき改善したといえる文の割合が、少なかったと言える。その原因としては、そもそも形態素分割に失敗している文であったため、改善策としては、置賜 Dict の拡張を行い、語彙を増やすことで、翻訳性能の改善が見られそうである。

F-score＝0.033～0.967 の文については全体的に Option+によって改善傾向にあるため、格助詞を補完する機能の性能を機械学習により向上させることや、第 1 章[1-4 置賜方言]にて述べている濁音化や促音便化という、その他の置賜方言の特徴を考慮した機能を方言翻訳システムに実装することにより正確な方言翻訳ができるようになると考える。

[BLEU-score による自動評価]

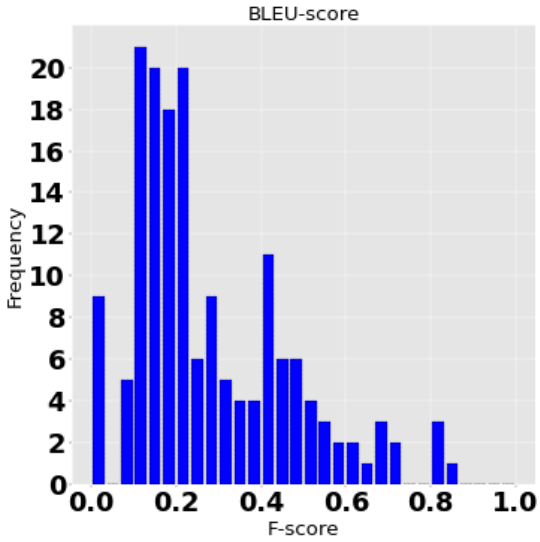

Fig.9-4　結果③の BLEU-score の分布

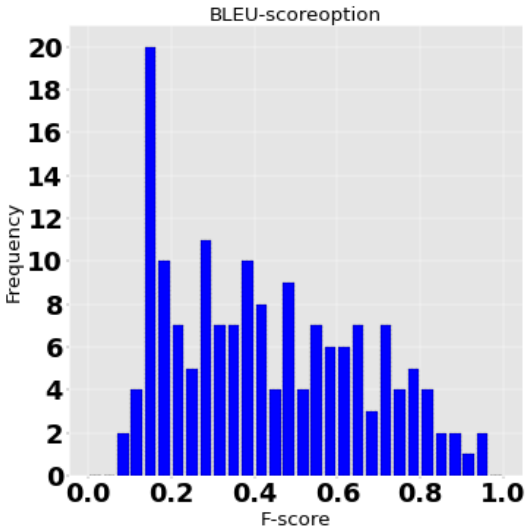

Fig.9-5　結果④の BLEU-score の分布

	結果③（ Base ）	結果④（Option+）	改善率
平均値	0.30	0.45	50%
中央値	0.21	0.41	95%

Fig.9-6　結果③・結果④から得られた BLEU-score の平均値・中央値と改善率

本実験では、4-gram について BLEU-score を算出し、平滑化手法として method7 を採用し実施した。結果より、Option+によって平均値・中央値共に大幅な改善がみられた。Option+により、正解文と翻訳文で同じ形態素が 4 つ並ぶ割合が増加したからだと考える。F-score 同様に置賜方言の特徴に沿って追加機能を実装することで BLEU-score の向上が見込める。今回は、最も人手評価に近いとされている method7 を採用したが、method0~6 の平滑化手法を採用し、置賜方言と共通語の翻訳自動評価において人手評価と最も近い手法を見つけることや、1-gram/2-gram/3-gram と他の n-gram について同様に実施するなど、置賜方言翻訳における BLEU-score の向上と評価方法の最適化が今後の課題である。

[人手による評価]

	文法性	置賜方言らしさ	意味の不変さ	正しい翻訳率[%]
結果③	2.49	2.22	1.74	32.9%
結果④	2.60	2.24	1.73	36.5%

Fig.9-7　結果③・結果④から得られた人手評価の平均値

格助詞の補完機能により、自動評価と人手評価の結果からそれぞれ精度が向上したといえる。正しい翻訳が行えている文が増えた一方で、本来格助詞がなくても意味が通じた文に、補完すべきでない格助詞が補完されてしまった文がいくつか見受けられた。そのため「意味の不変さ」を見ると格助詞の補完機能は、悪影響を受けていると言える。

[正解文に関する考察]

本稿の実験では、正解文を 1 つのみ人手にて作成し、分割評価・翻訳評価を行った。しかし意味が同じだが、単語が違うため、評価の数値では正解と評価されていない文が見られた。これらは本来、正解と評価されるべきである。以下の[正解文の例]のように、正解文を複数作成することで、人手評価により近い評価ができるようになるのではないかと考える。そのため、正解文を Chikkarpy と同義語辞書、同義語辞書の拡張により自動作成し、人手評価では正解とされる文が、自動評価でも同様に近い評価になるようにし、自動評価の有意性を向上させることが今後の課題である。

正解文の例：便所に行きたい / トイレに行きたい
※「便所」と「トイレ」は同じ意味であり、正解文を例のように 2 種類作ることで、正しいと判定される文の数が増えると考える。

[格助詞補完機能に関する考察]

格助詞が欠落していると判断する際、前後の品詞関係からのみ欠落を予測して、ランダムで補完を行った結果、F-score, BLEU-score, 人手評価のいずれの手法を用いた評価でも、同様に機械翻訳の性能は向上したといえる結果となった。一方、格助詞が省略されている場所を品詞の連続関係の条件によって格助詞を補完したため、本来格助詞がなくても意味が通じていた文章が、格助詞の補完によって意味が変化してしまった文章もあった。このようにランダムで補完を行うことは、一定数の性能向上がされたが、一部で性能低下の課題もあった。この問題への解決方法としては、機械学習の手法によるアプローチを合わせることによって、ランダムではなく品詞以外の要素の前後関係からの予測による格助詞補完の最適化を行い、翻訳性能の向上ができるのではないかと考える。

4-3. 類似研究との比較と考察

[分割再現率の比較]

本研究の分割再現率と広島方言・岡山方言の類似研究で算出されている再現率を比較した。本研究の再現率は、0.800 であり、比較対象は約 0.9 であった。再現率を約 0.9 にするには、置賜 Dict の量を増やすことで、再現率を向上させることができると考える。

置賜方言の分割再現率（ 本研究 ）	**0.800**
広島方言の分割再現率 [37]	**0.897**
岡山方言の分割再現率 [38]	**0.853 ~ 0.906**

[正しい翻訳率]

本研究の正しい翻訳率を同じ条件で人手評価を行った結果、関西弁の結果と比べ、近い数値が出ていることから、この結果は有効であると考える。

置賜方言-正しい翻訳率 人手評価（ 本研究 ）	**0.329**
関西弁-正しい翻訳率 人手評価 [23]	**0.286 ~ 0.403**

第5章. 結論

本研究では、介護現場で使われている置賜方言の例文を中心に、置賜方言を共通語へ変換するシステムを研究した。置賜 Dict の構築により形態素分割の性能が平均 28%向上し、機械翻訳による正解データとの精度・再現率からなる F 値は、0.55 となった。

現在の置賜 Dict の登録単語数は 1199 語であるが、置賜方言辞書[12]に収録されている語彙は 8000 語程度であるため、あと 7000 語ほど置賜 Dict に追加をすれば更なる精度向上が見込まれるが、全てに意味情報の付与を行うことは莫大なコストとなることが予想され、更にすべての動詞に活用情報を付与することは現実的ではない。本研究では、置賜方言翻訳システムの性能を向上させるための 1 つの施策として、格助詞をランダムに補完する手法を考案・実装し、F-score では平均 3%の向上がされ、BLEU-score では、平均 50%の改善がされ有効であった。

格助詞の補完については、前後の品詞関係からのみ欠落を予測して、ランダムで補完を行ったが、機械学習の手法によるアプローチを行うことで、ランダムではなく品詞以外の要素の前後関係からの予測による最適化が期待できる。第 2 章で述べた [2-3. 濁音化・促音便化・省略]について本研究では、頻繁に使われている語彙を辞書に登録することで、共通語への修正をおこなっていたが、発音する人によって濁音化などする基準が微細に異なるため、すべての語彙に対して、「濁音化・促音便化・省略」の有り得るすべての組み合わせを辞書へ追加することは困難であった。このような課題も含め、置賜方言の特徴を捉え、機械学習によるアプローチを行うことにより、低コストで方言翻訳の性能向上を目指すことができると考える。

評価手法では、人手評価と自動評価では正解とする判定方法に違いがあり、人手では意味が同じであれば単語が違う場合も高い評価となっていたが、自動評価では意味が同じでも単語が違うと低い評価となってしまう課題があった。Chikkarpy と同義語辞書による正解文の拡張を行い、自動で正解文を増やすことで、自動評価を人手評価に近づけることができるのではないかと考える。BLEU-score の平滑化手法の違いも検証しながら、人手評価により近づけた置賜方言における自動評価手法を模索することが今後の課題である。

本研究で作成した[形態素分割評価]・[翻訳システムの開発]・[翻訳評価]・[正解文]・[置賜 Dict]には、それぞれ改善点や課題があったが、類似研究と比較すると有効であるといえる結果が出た。今後は、これまでに述べた観点を含む様々な角度から、それぞれについてより深く注目することにより、介護現場などの実社会で有意な置賜方言翻訳システムができると考える。今後に期待したい。

[謝辞]
置賜方言ネイティブ話者としてご協力いただいた指導教員の横山道央准教授、形態素解析器 Sudachi に関する質問に答えていただいた株式会社ワークスアプリケーションズの開発者の皆様、置賜方言に関する情報収集にご協力いただいた株式会社ホリエの皆様、開発コードのレビューをしていただいた山形大学理学部の村上和隆さんに深く感謝いたします。

[参考文献]
[1] 厚生労働省　2025 年に向けた介護人材にかかる需給推計（確定値）について
[2] 厚生労働省老健局高齢者支援課,経済産業省製造産業局産業機械課, ロボット技術の介護利用における重点分野,平成 24 年 11 月制定 , 平成 29 年 10 月改定
[3] 後藤典子, 介護士が介護現場で使用している山形方言の特徴
[4] 後藤典子, 医療・介護現場の方言を外国人はどう理解するか ,日本語教育学会 161 巻 , 研究ノート, P42-P49
[5] 後藤典子, 医療・介護のための山形方言検索の工夫, 日本語教育方法研究会誌, Vol.22 No.1 , P60-P61
[6]公益社団法人国際厚生事業団,株式会社光洋スクエア,一般社団法人国際交流&日本語支援 Y「外国人のための 会話で学ぼう!介護の日本語 第 2 版: 指示がわかる、報告ができる」中央法規出版(2020-05-11)
[7] 方言が難しい都道府県ランキング　1 位に輝いたのは北か南か… ,
https://sirabee.com/2017/05/18/20161131919/ , 入手日 2021.12.06
[8]「方言」が難解すぎる都道府県ランキング ,
https://ranking.goo.ne.jp/column/6333/ , 入手日 2021.12.06
[9] 阿部香央莉,松林優一郎,岡崎直観,乾健太郎,ニュートラルネットを用いた多方言の翻訳と類型分析,自然言語処理学会発表論文集(2018 年 3 月)
[10]日高貢一郎(2007), シリーズ方言学 3 方言の機能, 岩波書店
[11] 平山輝男(1997),日本のことばシリーズ 6 山形県のことば,明治書院
[12] 菊池直, 読む方言百科事典　置賜のことば百科（上・下）笹原印刷
[13]SWING GIRLS おきたま応援サイト,おきたま弁講座,
http://swinggirls.jan.jp/okitama.html , 入手日 2021.12.06
[14] 山形県 しあわせ子育て応援部 しあわせ子育て政策課, 置賜地方・置賜弁, やまがた子育て応援サイト, https://kosodate.pref.yamagata.jp/dialect/okitama , 入手日 2021.12.06
[15] おきたま弁講座, SWING GIRLS おきたま応援サイト,
http://swinggirls.jan.jp/okitama.html , 入手日 2021.12.06
[16] 投稿米沢の方言, http://ayrtonsports.com/hougen/hougen.html 入手日 2021.12.06
[17] ひとりごとダイアリー・アーカイブス,方言・米沢弁・米沢でしゃべらっちぇきた言葉集　おきたまのラジオマン編 , https://okitama-npo.sakura.ne.jp/, 入手日 2021.12.06

[18] 米沢弁 方言集 691 ワード, http://www.omn.ne.jp/~yamada-s/hoogen/yonezawa.htm, 入手日 2021.12.06
[19] Sudachi: a Japanese Tokenizer for Business, Kazuma Takaoka, Sorami Hisamoto, Noriko Kawahara, Miho Sakamoto, Yoshitaka Uchida and Yuji Matsumoto
[20] Large Scale Dictionary Development for Sudachi ,Proceedings of Language Resources Workshop ,Miho SAKAMOTO,Noriko KAWAHARA,Sorami HISAMOTO,Kazuma TAKAOKA,Yoshitaka UCHIDA
[21]WorksApplications , https://github.com/WorksApplications/ , 入手日 2021.12.06
[22] 工藤拓(2018) 「実戦・自然言語処理シリーズ 第 2 巻 形態素解析の理論と実装」 近代科学社
[23] 長谷川 駿 他, 事前学習と汎化タグによる方言翻訳の精度向上, 情報処理学会 自然言語処理研究会 研究報告 Vol.2017-NL-233 No.2
[24] [Kishore Papineni, Salim Roukos and Todd Ward, et al, BLEU: a Method for AutomaticEvaluationofMachineTranslation,ACL'02](https://www.aclweb.org/anthology/P02-1040.pdf)
[25] A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU (http://acl2014.org/acl2014/W14-33/pdf/W14-3346.pdf)
[26] 分析ノート MeCab に実装されている最小コスト法について (https://analytics-note.xyz/programming/mecab-cost/ s)
[27] 形態素解析とは？おすすめの 5 大解析ツールや実際の応用例を紹介 (https://udemy.benesse.co.jp/data-science/ai/morphological-analysis.html)
[28] 複数粒度の分割結果に基づく日本語単語分散表現, 株式会社ワークスアプリケーションズ, 人間文化研究機構 国立国語研究所 , 言語処理学会 第 25 回年次大会 発表論文集
[29] 形態素解析器 Sudachi の「辞書」はどのように作られているか: 複数の分割単位を例として, (https://zenn.dev/sorami/articles/c9a506000fd1fbd1cf98)
[30] Smoothly BLEU,(https://qiita.com/taxio/items/ccf1ddba2214b759e636)
[31] Natural Language Toolkit, (http://nltk.org/) , Chin Yee Lee, Hengfeng Li, Ruxin Hou, Calvin Tanujaya Lim
[32] 柴田直由 他, 年代差を考慮した方言翻訳システム, 平成 22 年度第 6 回情報処理学会東北支部研究会, 資料番号 10-6-B33
[33] 佐藤守 他, 方言から共通語への翻訳システムに関する基礎的研究 , 自然言語処理研究会報告 150,85-90
[34] 横山昌一 他, 共通語から山形村山方言への翻訳システム, 言語処理学会第 13 回年次大会, pp.923-926
[35] 横山昌一 他,機械処理のための方言結合価辞書の作成 –置賜方言を対象に ,言語処理学会 第 4 回年次大会 発表論文集 pp286 - 289

[36] 佐藤守 他, 山形方言から共通語への翻訳システム , 言語処理学会 第 9 回年次大会 発表論文集 401 - 404
[37] 中本典子 他, 広島弁音声認識のためのコーパスと言語モデルの構築, 言語処理学会 第 18 回年次大会 発表論文集(2012 年 3 月） pp 883 – 886
[38] 福圓 琢真 他, 標準語用のパラメータを流用した岡山弁向け統計的形態素解析, FIT2015（第 14 回情報科学技術フォーラム）pp 209-210 第二分冊

付録資料

付録資料 1. 注釈付き置賜方言コーパス（置賜 Dict）
付録資料 2. 実験用置賜方言例文コーパス
付録資料 3. 実験に使用したプログラミングコード
付録資料 4. 実験によって得られた結果表

付録資料 1　注釈付き置賜方言コーパス（置賜 Dict）

注釈付き置賜方言コーパス(置賜 Dict)は、大規模なデータのため、先頭 10 行のみ本稿に掲載し、以降はインターネットにて公開する。所属研究室の関係者のみ閲覧可能な Notion と執筆者の GitHub にて招待者限定公開を行う。

Notion
横山・原田研究室/介護支援[自然言語処理]/二瓶 file/卒業論文-付録資料
GitHub
https://github.com/nihei1206/OitamaDict

置賜弁テスト	1	1	-32768	見出し結果表示	品詞 1	品詞 2	品詞 3	品詞 4	品詞 9(活用形)	品詞 10(活用形)	ヨミ	正規化表記	0	*	*	*	*
あいしょらう	1286	1286	-32768	あいしょらう	動詞	五段			を	方言	アイショラウ	あしらう	1	*	*	*	*
あいべ	1286	1286	-32768	あいべ	動詞	不規則			へ	方言	アイベ	行こう	2	*	*	*	*
あえしょらう	1286	1286	-32768	あえしょらう	動詞	五段			を	方言	アイショラウ	あしらう	3	*	*	*	*
あえべ	1286	1286	-32768	あえべ	動詞	不規則			へ	方言	アエベ	行こう	4	*	*	*	*
あがえ	1286	1286	-32768	あがえ	動詞	一般			を	方言	アガエ	食べなさい	5	*	*	*	*
あがっておごやえ	1286	1286	-32768	あがっておごやえ	動詞	不変則			を	方言	アガッテオゴヤエ	食べてください	6	*	*	*	*

あがら	1286	1286	-32768	あがら	動詞	下一段			を	方言	アガラ	食べ	7	*	*	*	*
あがり	1286	1286	-32768	あがり	動詞	下一段			を	方言	アガリ	食べ	8	*	*	*	*
あがる	1286	1286	-32768	あがる	動詞	下一段			を	方言	アガル	食べる	9	*	*	*	*
あがる	1286	1286	-32768	あがる	動詞	下一段			を	方言	アガル	食べる	10	*	*	*	*

付録資料 2　実験用置賜方言例文コーパス

実験用置賜方言例文コーパスは、大規模なデータのため、先頭 10 行のみ本稿に掲載し、以降についてはインターネットにて公開する。所属研究室の関係者のみ閲覧可能な Notion と研究室のハードディスク、執筆者の GitHub にて招待者限定公開を行う。

Notion
横山・原田研究室/介護支援[自然言語処理]/二瓶 file/卒業論文-付録資料
GitHub
https://github.com/nihei1206/OitamaDict/blob/main/auto_evaluation.csv

標準語	置賜弁	正解分割
あの人、何をしているの	あいづ、何しったなや	あいづ,、.何,しっ,た,な,や
あれ、取ってください	あいづ、取ってけろ	あいづ,、,取っ,て,けろ
かかとが痛い	あぐどいだい	あぐど,いだい
かかとが痛くないですか	あぐどいだぐねぇが	あぐど,いだ,ぐ,ねぇ,が
頭がとても痛い	あだまかなりやめる	あだま,かなり,やめる
頭が痛くて持て余している	あだまやめでしゃーましする	あだま,やめ,で,しゃーまし,する
頭が痛い	あだまやめる	あだま,やめる
暑くないですか	あづぐねえが	あづく,ねえ,が
暖かいお茶をください	あったけぇお茶くだい	あったけぇ,お茶,くだい
あとで看護師に診てもらいましょう	あどで看護師に診てもらうべ	あど,で,看護,師,に,診,て,もらう,べ

付録資料 3　実験に使用したプログラミングコード

プログラミングコードは、執筆者の GitHub にて公開した。

GitHub

https://github.com/nihei1206/oitama_trans

分割性能自動評価システムのプログラミングコード

oitama_trans/oitama_system/hyoka/hyoka_split.py /

```
from sudachipy import tokenizer as t
from sudachipy import dictionary as d

def sudachionlyWakachi(text:str) -> list:
    """
    sudachi_dict だけでの分割
    """
    hyokaArrayonly = []
    mode = t.Tokenizer.SplitMode.A
    config_path_link = "../lib/python3.9/site-
packages/sudachipy/resources/notuse_resources/sudachi.json"
    tokenizer_obj = d.Dictionary(config_path=config_path_link,dict="full").create()
    tokens = tokenizer_obj.tokenize(text,mode)
    for m in tokens:
        hyokaArrayonly.append(m.surface())
    return hyokaArrayonly

def sudachiOitamaWakachi(text:str) -> list:
    """
    sudachi_dict と Oitama_dict での分割
    """
    outputArray = []
    mode = t.Tokenizer.SplitMode.A
    config_path_link = "../lib/python3.9/site-packages/sudachipy/resources/sudachi.json"
    tokenizer_obj = d.Dictionary(config_path=config_path_link,dict="full").create()
    tokens = tokenizer_obj.tokenize(text,mode)
    for m in tokens:
        outputArray.append(m.surface())
    return outputArray

def hyoka(result:list,answer:list) -> list:
    """
    fig.8-1:
    return result,answer,precision,recall,Fscore
    """
    correct = 0
    result_index = 0
    answer_index = 0
    result_pos = 0
    answer_pos = 0
```

```
    while result_index < len(result) and answer_index < len(answer):
        if result_pos == answer_pos:
            if result[result_index] == answer[answer_index]:
                correct += 1
            result_pos += len(result[result_index])
            answer_pos += len(answer[answer_index])
            result_index += 1
            answer_index += 1

        elif result_pos > answer_pos:
            answer_pos += len(answer[answer_index])
            answer_index += 1

        elif result_pos < answer_pos:
            result_pos += len(result[result_index])
            result_index += 1

    precision = correct / len(result)
    recall = correct / len(answer)

    if precision == 0 and recall == 0:
        return result,answer,precision ,recall ,0
    else:
        Fscore = (2 * precision * recall) / ( precision + recall )
        # return result,answer,precision,recall,Fscore
        return result,answer,precision ,recall ,Fscore

def canmaBunkatsu(text:str) -> list:
    """
    ,分割
    """
    outputArray = text.split(',')
    return outputArray
```

置賜方言翻訳システムのプログラミングコード

oitama_trans/oitama_system/translator/translate.py

```
from sudachipy import tokenizer as t
from sudachipy import dictionary as d
from translator import option as op

### Initialized Takenizer ###
# トークナイザの作成,辞書位置(sudachi.json ; 相対パス)の指定
# Qsudachidict_full 優先  dict="full"

def replacement(text:str,option:int):
    """
    #入力された文章を,SudachiDict と UserDict で翻訳
    """
    #ユーザー辞書のパスを宣言
    config_path_link = "../lib/python3.9/site-packages/sudachipy/resources/sudachi.json"
    #ユーザー辞書の使用を宣言
```

```
    tokenizer_obj = d.Dictionary(config_path=config_path_link).create()
    mode = t.Tokenizer.SplitMode.C
    combinedExchangeHogen = []
    combinedExchangeHogentext = ''
    tokens = tokenizer_obj.tokenize(text,mode)
    for m in tokens:
        if m.part_of_speech()[5] == '方言':
            combinedExchangeHogen.append(m.normalized_form())
        else:
            combinedExchangeHogen.append(m.surface())

    if option == 0:
        combinedExchangeHogentext = "".join(combinedExchangeHogen)
        return combinedExchangeHogentext
    elif option == 1:
        #欠落の格助詞補完
        combinedExchangeHogentext = "".join(combinedExchangeHogen)
        translatedOutput = op.addDropedWord(combinedExchangeHogentext)
        return translatedOutput
```

格助詞補完機能に関するプログラミングコード

oitama_trans/oitama_system/translator/option.py /

```
from sudachipy import tokenizer as t
from sudachipy import dictionary as d
import random
from statistics import mean
from hyoka import hyoka_trans as ht

def addDropedWord(text:str) -> str:
    """
    欠落語補完機能
    名詞,形容詞が連続した場合,間に「が」を入れることで「が」の欠落を補完
    名詞,動詞が連続した場合,間に「に」を入れることで「に(さ)」の欠落を補完
    仮説 ->「さ」助詞を補完したらなんでもうまく行くのではないか
    口語文書の解析精度向上のための 助詞落ち推定および補完 ...の論文より割合を引用
    default = {'が':15.2,'を':16.7,'に':0.7,'で':1.4,'の':1.5,'は':31.5,'と':33.0}
    置賜カスタマイズ(「山形県のことば」より、省略される助詞と明記されているもののみを実行)
    yamagata custom = {'が':23.42,'を':25.73,'の':2.31,'は':48.53} ->4kjs
    {'が':71.95,'を':25.73,'の':2.32} -> 3kjs
    """
    config_path_link = "../lib/python3.9/site-packages/sudachipy/resources/sudachi.json"
    tokenizer_obj = d.Dictionary(config_path=config_path_link).create()
    mode = t.Tokenizer.SplitMode.C
    tokens = tokenizer_obj.tokenize(text,mode)
    wordPartofspeech = []
    wordCombine = []
    kakujoshi_dict = {'が':1,'を':1,'に':1}
    candidates = [*kakujoshi_dict]
    weights = [*kakujoshi_dict.values()]
    randomed_kakujoshi = random.choices(candidates, weights=weights)[0]
```

```
    for m in tokens:
        wordPartofspeech.append([m.surface(),m.part_of_speech()[0]])

    for j in range(len(wordPartofspeech)-1):
        if wordPartofspeech[j][1] == '名詞' or wordPartofspeech[j][1] == '代名詞':
            if wordPartofspeech[j+1][1] == '動詞' or wordPartofspeech[j+1][1] == '形容詞
':
                wordPartofspeech.insert(j+1,[randomed_kakujoshi,'助詞',''])

    for k in range(len(wordPartofspeech)):
        wordCombine.append(wordPartofspeech[k][0])
    wordOutput = "".join(wordCombine)
    return wordOutput

def manytimeFscore(text1:str,text2:str,n:int):
    """
    何度もランダム格助詞補完法を実施し、その平均を取る
    一番精度が高かった値を return
    """
    arr = []
    for k in range(1,n+1):
        if not ht.translate_hyoka(text1,text2,1)[3] == None:
            arr.append(ht.translate_hyoka(text1,text2,1)[3])
    max_ = [i for i, v in enumerate(arr) if v == max(arr)]
    maxtimesIndex = int(mean(max_))
    maxArr = float(max(arr))
    return maxArr,maxtimesIndex
```

翻訳評価システムのプログラミングコード

oitama_trans/oitama_system/hyoka/hyoka_trans.py /

```
from sudachipy import tokenizer as t
from sudachipy import dictionary as d
# from nltk.util import ngrams
from nltk import bleu_score as bs
from translator import translate

def hyokaArray_trans(text:str) -> list:
    """
    #入力された文章を,評価するための配列に sudachidict のみでわかち書き
    #splitMode == A
    """
    ### Initialized Takenizer ###
    mode = t.Tokenizer.SplitMode.A
    config_path_link = "../lib/python3.9/site-
packages/sudachipy/resources/notuse_resources/sudachi.json"
    tokenizer_obj = d.Dictionary(config_path=config_path_link,dict="full").create()
    hyokaArray = []
    tokens = tokenizer_obj.tokenize(text,mode)
    for m in tokens:
```

```
            hyokaArray.append(m.surface()+str(m.part_of_speech()))
    return hyokaArray
    # surface()+part_of_speech() -> (表層形 + 品詞情報[例:動詞/五段活用/さ変可能])

def list_difference(list1:list, list2:list) -> list:
    """
    #配列差分出力関数
    #list1 にあって、list2 にない要素を出力
    #list1 に重複{2 つ同じ要素}があったら、2 つとカウント。
    #list2 に同一要素が 1 つだとしたら、1 つだけ出力される
    """
    result = list1.copy()
    for value in list2:
        if value in result:
            result.remove(value)
    return result

def translate_hyoka(oitama:str, answer:str ,option:int) -> list:
    """
    hyokaOption == 0 ->置換のみ手法
    hyokaOption == 1 -> 脳筋 Option(格助詞の確立的はめ込み)
    #return [answer:str,oitama:str,result:str,fScore:float,bleuScore:float]
    """
    #置賜弁をここで標準語に翻訳
    result = translate.replacement(oitama,option)

    #result:置賜弁を翻訳した結果
    #answer:用意している正解データ
    resultHyokaArray = hyokaArray_trans(result)
    answerHyokaArray = hyokaArray_trans(answer)

    # 正解データをわかち書き
    # 正解文を形態要素解析した正解配列と翻訳したあとの文を形態要素解析した配列を比
較して、
    # 正解分にしかない配列の要素を配列にして出力したのが Only_answer_list_have
    Only_answer_list_have = list_difference(answerHyokaArray,resultHyokaArray)
    system_output_word_count = len(resultHyokaArray) #翻訳した結果の全単語数
    answer_output_word_count = len(answerHyokaArray) #対応する正解データの全単語数
    correct_output_word_count = answer_output_word_count-len(Only_answer_list_have)

    seido = 0
    saigen = 0

    try:
        seido = correct_output_word_count/system_output_word_count
        saigen = correct_output_word_count/answer_output_word_count
    except ZeroDivisionError as e:
        print(e)
        print(type(e))

    ### Output ###
    # F-score
    # print("-- F score --")
    # print((2*seido*saigen)/(seido+saigen))
```

```
    if seido+saigen == 0:
        fScore = 0
    else:
        fScore = (2*seido*saigen)/(seido+saigen)

    # BLEU-score used option mothod7 (smoothing Function)
    # print("-- BLEU score --")
    #
print(bleu_score.sentence_bleu([resultHyokaArray],answerHyokaArray,smoothing_function=
bleu_score.SmoothingFunction().method7))
    # weights = (1./5., 1./5., 1./5., 1./5., 1./5.)
    # ngram の重み->weights これは連続する token の比較を行うんだけど、その n 連続と
いう意味
    bleuScore =
bs.sentence_bleu([resultHyokaArray],answerHyokaArray,smoothing_function=bs.Smoothing
Function().method7)

    if bleuScore == 0:
        bleuScore = None
    # input = (oitama:str , answer:str)

    return [oitama,result,answer,fScore,bleuScore]
```

結果を CSV 出力するためのプログラミングコード

oitama_trans/oitama_system/make_csv.py /

```
import time
import csv
import datetime
from tqdm import tqdm
import numpy as np
from statistics import mean
from translator import translate as tr,option as op
from hyoka import hyoka_trans as ht,hyoka_split as hs

def average_output(array:list,option:int) -> list:
    # option 0 => for split evaluate CSV
    if option == 0:
        seido_arr = []
        saigen_arr = []
        fscore_arr = []
        for i in range(len(array)):
            seido_arr.append(array[i][2])
            saigen_arr.append(array[i][3])
            fscore_arr.append(array[i][4])
        seido_ave = mean(seido_arr)
        saigen_ave = mean(saigen_arr)
        f_ave = mean(fscore_arr)
        return '平均',None,seido_ave,saigen_ave,f_ave

    if option == 1:
        # option 1 => for translate evaluate CSV
        numarr = np.array(array).transpose()
```

```
        f_ave = mean(numarr[3])
        f_op_ave = mean(numarr[6])
        index_ave = mean(numarr[7])
        # bleu_ave = mean(numarr[4])
        return '平均',None,None,f_ave,None,None,f_op_ave,index_ave


def make_csv_split():
    '''
    '''

    start = time.time()
    with open('../auto_evaluation.csv') as f:
        reader = csv.reader(f)
        inputArray = [row for row in reader]
    f.close()

    outputArray = []
    for i in tqdm(range(1,len(inputArray))):
        np.pi*np.pi

outputArray.append(hs.hyoka(hs.sudachiOitamaWakachi(inputArray[i][1]),hs.canmaBunkats
u(inputArray[i][2])))

    outputArray.append(average_output(outputArray,0))

    header = ['result','answer','seido','saigen','Fscore']
    dt_now = datetime.datetime.now()
    with open('../outputCSV/Bunkatsu_OitamaDict'+
dt_now.strftime('%y%m%d-%H%M%S') +'.csv', 'w') as f:

        writer = csv.writer(f)
        writer.writerow(header)
        writer.writerows(outputArray)

    f.close()

    outputArray = []
    for i in tqdm(range(1,len(inputArray))):
        np.pi*np.pi

outputArray.append(hs.hyoka(hs.sudachionlyWakachi(inputArray[i][1]),hs.canmaBunkatsu(i
nputArray[i][2])))

    outputArray.append(average_output(outputArray,0))
    header = ['result','answer','seido','saigen','Fscore']
    dt_now = datetime.datetime.now()
    with open('../outputCSV/Bunkatsu_sudachiDict'+
dt_now.strftime('%y%m%d-%H%M%S') +'.csv', 'w') as f:

        writer = csv.writer(f)
        writer.writerow(header)
        writer.writerows(outputArray)
```

```
    f.close()
    elapsed_time = time.time() - start

    return print("elapsed_time:{0}".format(elapsed_time) + "[sec]")

def make_csv_translate():
    """

    """
    print('何回ぶん回しますか?')
    n_time = int(input())

    start = time.time()
    with open('../auto_evaluation.csv') as f:
        reader = csv.reader(f)
        inputArray = [row for row in reader]
    f.close()

    outputArray = []
    for i in tqdm(range(1,len(inputArray))):
        np.pi*np.pi
        adMF = ht.translate_hyoka(inputArray[i][1],inputArray[i][0],0)
        adMF.append(tr.replacement(inputArray[i][1],1))
        #  ぶん回す↓
        manytime = op.manytimeFscore(inputArray[i][1],inputArray[i][0],n_time)
        adMF.append(manytime[0])
        adMF.append(manytime[1])
        outputArray.append(adMF)
        # np.pi*np.pi
        # adMF = translate_hyoka(inputArray[i][1],inputArray[i][0],0)
        # adMF.append(translaterOitamaOption(inputArray[i][1],tokenizer_obj))


    outputArray.append(average_output(outputArray,1))

    # for_plot_array
    header =
['oitama','result','answer','fScore','bleuscore','option_result','opt_Arr_max','maxtimesIndex']
    dt_now = datetime.datetime.now()
    with open('../outputCSV/OitamaTrans'+ dt_now.strftime('%y%m%d-%H%M%S')
+'.csv', 'w') as f:

        writer = csv.writer(f)
        writer.writerow(header)
        writer.writerows(outputArray)

    f.close()
    elapsed_time = time.time() - start
    return print("elapsed_time:{0}".format(elapsed_time) + "[sec]")
```

結果の CSV からヒストグラムを作図するためのプログラミングコード

oitama_trans/oitama_system/make_figure.py /

```
import csv
import datetime
import numpy as np
import matplotlib.pyplot as plt
import statistics as st


def makeFig(array, data_name:str,bins:int,sub_name,title:str,dataType):
	median = st.median(array)
	ave = st.mean(array)
	plt.style.use('ggplot')
	plt.figure(figsize=(10, 10), dpi=40)
	plt.hist(array,bins=bins,range = (0,1), color="blue", edgecolor="black", linestyle="--
",rwidth = 0.8)
	plt.title(title, fontsize = 24,color = 'k')
	plt.xticks(fontsize = 32,color = 'k', fontweight = 'bold')
	plt.xlabel("F-score", fontsize=24,color = 'k')
	plt.ylabel("Frequency", fontsize=24,color = 'k')

	if dataType == 's':
		plt.yticks(np.arange(0, 65, 5),fontsize=32, fontweight = 'bold',color = 'k')
	if dataType == 't':
		plt.yticks(np.arange(0, 22, 2),fontsize=32, fontweight = 'bold',color = 'k')

	dt_now = datetime.datetime.now()
	plt.savefig("../outputFig/" +str(data_name) + str(sub_name)+
dt_now.strftime('%H%M%S'))
	plt.show()

	return "Done! " + str(data_name) + str(sub_name)+ ": median is " + str(median) +    " /
Ave is " + str(ave)

def selectFigType():
	"""
	図を作りたい
	"""

	print('split(s) or trans(t)')
	dataType = str(input())
	if dataType == 's':
		print('Oitama or Sudachi')
		dataname = str(input())
	elif dataType == 't':
		dataname = str('Translator')
	print('input bins')
	bins = int(input())
	print('title of histgram')
	title = str(input())
	print('input import file name')
	imported_csvname = str(input())
```

```
with open("../outputCSV/" + imported_csvname) as f:
    reader = csv.reader(f)
    inputArray = [row for row in reader]
f.close()

inputArray = np.array(inputArray[1:-1])

if dataType == 'split' or   dataType == 's':
    floatArray = np.array(inputArray.transpose(1, 0)[4],dtype = np.float64)
    print(makeFig(floatArray,dataname,bins,title,dataType))
elif dataType == 'trans' or   dataType == 't':
    floatArray = np.array(inputArray.transpose(1, 0)[3],dtype = np.float64)
    print(makeFig(floatArray,dataname,bins,title,dataType))

    floatArray = np.array(inputArray.transpose(1, 0)[6],dtype = np.float64)
    print(makeFig(floatArray,dataname,bins,title,dataType))
```

付録資料 4　実験によって得られた結果表

実験用置賜方言例文コーパスは、大規模なデータのため、先頭 10 行のみ本稿に掲載し、以降についてはインターネットにて公開する。所属研究室の関係者のみ閲覧可能な Notion と執筆者の GitHub にて招待者限定公開を行う。

Notion

横山・原田研究室/介護支援[自然言語処理]/二瓶 file/卒業論文-付録資料

GitHub

https://github.com/nihei1206/OitamaDict/blob/main/auto_evaluation.csv

分割性能自動評価の結果表

Sudachi Dict のみによる分割

result	answer	seido	saigen	Fscore
['あいづ', '、', '何', 'しっ', 'たな', 'や']	['あいづ', '、.何', 'しっ', 'た', 'な', 'や']	0.16666666666666700	0.16666666666666700	0.16666666666666700
['あいづ', '、', '取っ', 'て', 'けろ']	['あいづ', '、', '取っ', 'て', 'けろ']	1.0	1.0	1.0
['あぐど', 'いだい']	['あぐど', 'いだい']	1.0	1.0	1.0
['あ', 'ぐ', 'どい', 'だ', 'ぐ', 'ねぇ', 'が']	['あぐど', 'いだ', 'ぐ', 'ねぇ', 'が']	0.42857142857142900	0.6	0.5
['あ', 'だ', 'ま', 'かなり', 'やめる']	['あだま', 'かなり', 'やめる']	0.4	0.6666666666666670	0.5
['あだま', 'やめ', 'で', 'しゃーまし', 'する']	['あだま', 'やめ', 'で', 'しゃーまし', 'する']	1.0	1.0	1.0

['あ', 'だ', 'ま', 'やめる']	['あだま', 'やめる']	0.25	0.5	0.3333333333333330
['あ', 'づ', 'ぐ', 'ね', 'え', 'が']	['あづぐ', 'ねえ', 'が']	0.16666666666666700	0.3333333333333330	0.2222222222222220
['あっ', 'たけ', 'ぇ', 'お茶', 'ください']	['あったけぇ', 'お茶', 'ください']	0.4	0.6666666666666670	0.5
['あ', 'ど', 'で', '看護', '師', 'に', '診', 'て', 'もらう', 'べ']	['あど', 'で', '看護', '師', 'に', '診', 'て', 'もらう', 'べ']	0.8	0.8888888888888890	0.8421052631578950

Sudachi Dict と置賜 Dict による分割

result	answer	seido	saigen	Fscore
['あいづ', '、', '何', 'しっ', 'たな', 'や']	['あいづ', '、.何', 'しっ', 'た', 'な', 'や']	0.16666666666666700	0.16666666666666700	0.16666666666666700
['あいづ', '、', '取っ', 'て', 'けろ']	['あいづ', '、', '取っ', 'て', 'けろ']	1.0	1.0	1.0
['あぐど', 'いだい']	['あぐど', 'いだい']	1.0	1.0	1.0
['あ', 'ぐ', 'どい', 'だ', 'ぐ', 'ねぇ', 'が']	['あぐど', 'いだ', 'ぐ', 'ねぇ', 'が']	0.42857142857142900	0.6	0.5
['あ', 'だ', 'ま', 'かなり', 'やめる']	['あだま', 'かなり', 'やめる']	0.4	0.6666666666666670	0.5
['あだま', 'やめ', 'で', 'しゃーまし', 'する']	['あだま', 'やめ', 'で', 'しゃーまし', 'する']	1.0	1.0	1.0
['あ', 'だ', 'ま', 'やめる']	['あだま', 'やめる']	0.25	0.5	0.3333333333333330
['あ', 'づ', 'ぐ', 'ね', 'え', 'が']	['あづぐ', 'ねえ', 'が']	0.16666666666666700	0.3333333333333330	0.2222222222222220
['あっ', 'たけ', 'ぇ', 'お茶', 'ください']	['あったけぇ', 'お茶', 'ください']	0.4	0.6666666666666670	0.5
['あ', 'ど', 'で', '看護', '師', 'に', '診', 'て', 'もらう', 'べ']	['あど', 'で', '看護', '師', 'に', '診', 'て', 'もらう', 'べ']	0.8	0.8888888888888890	0.8421052631578950

翻訳性能自動評価の結果表

oitama	result	answer	fScore	bleuscore	option_result	opt_Arr_max	maxtimesIndex
あいづ、何しったなや	あれ、何しったや	あの人、何をしているの	0.2857142857142860	0.14154751160645100	あれ、何しったや	0.2857142857142860	14
あいづ、取ってけろ	あれ、取ってください	あれ、取ってください	1.0	1.1167470964180200	あれ、取ってください	1.0	14

あぐどいだい	かかと痛い	かかとが痛い	0.8	0.19992254095048100	かかとに痛い	1.0	17
あぐどいだぐねぇが	あぐどいたぐないか	かかとが痛くないですか	0.3333333333333330	0.133051885050407	あぐどをいたぐをないか	0.42857142857142900	15
あだまかなりやめる	あたまかなり痛い	頭がとても痛い	0.25	0.1394721495522780	あたまかなりに痛い	0.4444444444444450	13
あだまやめでしゃーましする	頭やめで持て余しする	頭が痛くて持て余している	0.42857142857142900	0.1677888457629700	頭やめで持て余しする	0.42857142857142900	14
あだまやめる	あたま痛い	頭が痛い	0.4	0.15587075056736400	あたまが痛い	0.6666666666666670	12
あづぐねえが	あづぐないえか	暑くないですか	0.2	0.08459413487948230	あづぐないえか	0.2	14
あったけぇお茶くだい	あったけぇお茶ください	暖かいお茶をください	0.20000000000000000	0.12114911497845300	あったけぇお茶をください	0.3636363636363640	13
あどで看護師に診てもらうべ	あどで看護師に診てもらうだ	あとで看護師に診てもらいましょう	0.631578947368421	0.609365237041581	あどで看護師に診てもらうだ	0.631578947368421	14

人手評価の結果表

方言翻訳システム Base による結果の人手評価

元の置賜方言	システム出力結果	文法性	共通語らしさ	意味の不変さ	用意した正解データ
あいづ、何しったなや	あれ、何しったや	2	1	変化している	あの人、何をしているの
あいづ、取ってけろ	あれ、取ってください	3	3	変化していない	あれ、取ってください
あぐどいだい	かかと痛い	2	3	変化していない	かかとが痛い
あぐどいだぐねぇが	あぐどいたぐないか	1	1	変化している	かかとが痛くないですか
あだまかなりやめる	あたまかなり痛い	3	3	変化していない	頭がとても痛い

あだまやめでしゃーまする	頭やめで持て余しする	1	1	変化している	頭が痛くて持て余している
あだまやめる	あたま痛い	3	3	変化していない	頭が痛い
あづぐねえが	あづぐない家か	1	1	変化している	暑くないですか
あったけぇお茶くだい	あったけぇお茶ください	3	3	変化していない	暖かいお茶をください
あどで看護師に診てもらうべ	あどで看護師に診てもらうだ	2	2	変化していない	あとで看護師に診てもらいましょう

方言翻訳システム Option による結果の人手評価

元の置賜方言	格助詞補完_出力結果_opNWG	文法性	共通語らしさ	意味の不変さ	用意した正解データ
あいづ、何しったなや	あれ、何しったや	2	1	変化している	あの人、何をしているの
あいづ、取ってけろ	あれ、取ってください	3	3	変化していない	あれ、取ってください
あぐどいだい	かかとが痛い	3	3	変化していない	かかとが痛い
あぐどいだぐねぇが	あぐどをいたぐをないか	2	1	変化している	かかとが痛くないですか
あだまかなりやめる	あたまかなりに痛い	3	3	変化していない	頭がとても痛い
あだまやめでしゃーまする	頭やめで持て余しする	1	1	変化している	頭が痛くて持て余している
あだまやめる	あたまに痛い	3	3	変化していない	頭が痛い
あづぐねえが	あづぐない家か	1	1	変化している	暑くないですか
あったけぇお茶くだい	あったけぇお茶をください	3	3	変化していない	暖かいお茶をください
あどで看護師に診てもらうべ	あどで看護師に診てもらうだ	2	2	変化していない	あとで看護師に診てもらいましょう