令和 4 年度茨城大学工学部情報工学科

卒業研究論文

CrossWeigh の日本語 NER データセットへの適用とラベルノイズの調査

所属　情報工学科

著者　西村柾人 (19T4054A)

指導教員　新納浩幸教授

令和 5 年 2 月 3 日 (金)

令和 4 年度茨城大学工学部情報工学科　卒業研究論文

CrossWeigh の日本語 NER データセットへの適用とラベルノイズの調査

著者

西村柾人 (19T4054A)

指導教員

新納浩幸教授

論文要旨

通常, 教師あり学習は訓練データには誤りがないという前提で学習が行われるが, 実際には誤りを含む場合も多い. 特に固有表現認識のタスクに使われるデータセットには、ラベルの定義に曖昧なものがあるため, タグ付けには誤りが生じやすい. これらの誤りはラベルノイズと呼ばれ、大量のデータをモデルの学習に利用する深層学習において、間違いであるラベルに対して過学習を行ってしまい、モデルの性能劣化につながってしまうという問題が報告されている。

このような問題を解決するために最近では、DCAI という使用するデータセットをより正確なものにすることで、モデルの精度を改善するという考え方が広まっている。

これらの背景から Wang らは、使用するデータセット内にラベルノイズが含まれていたとしても、それらのラベルノイズがモデルの学習時に与える悪影響を小さくする効果を持ったモジュールである CrossWeigh を提案した。この CrossWeigh を利用することで、同じデータセットで学習を行った機械学習モデルを用いた英語の固有表現認識のタスクにおいて、精度向上に成功している.

本稿では Wang らが提案した CrossWeigh を日本語の固有表現認識のデータセットに適用し, CrossWeigh が日本語においてもモデルの精度向上に貢献するのかどうかを確認する. 同時に, CrossWeigh によって生成される重み付きデータを調査することによってデータセット内にあるラベルノイズの発見を試みた.

さらに、それらの発見したラベルノイズを修正した学習データと削除した学習データを作成して、NER モデルに学習させることで、それらの行為がモデルの識別精度に貢献するのかについても調査した。

目次

表目次

図目次

第 1 章

序論

固有表現認識 (NER; Named Entity Recognition) とは、トークナイズされた文章をモデルに入力し、そのトークンごとにふさわしい固有表現 (NE; Named Entity) を出力させるタスクである。

図 1.1 を例にすると、入力として「茨城大学は茨城県にある大学です。」という文章をトークンごとに分割し、モデルに入力することで、それぞれのトークンに対応した施設名や地名などのラベルが付与される。また、固有表現でないトークンにはそれを意味する O ラベルが付与される。

近年では, BERT [1] などに代表される学習データを大量に用意し, モデルを学習させる機械学習手法が注目を浴びている. しかし, この学習データ内のラベルにノイズが含まれている場合, 深層学習を利用したこれらの手法では, ラベルノイズに対して過学習してしまい, 性能劣化につながるという問題が指摘されている [2].

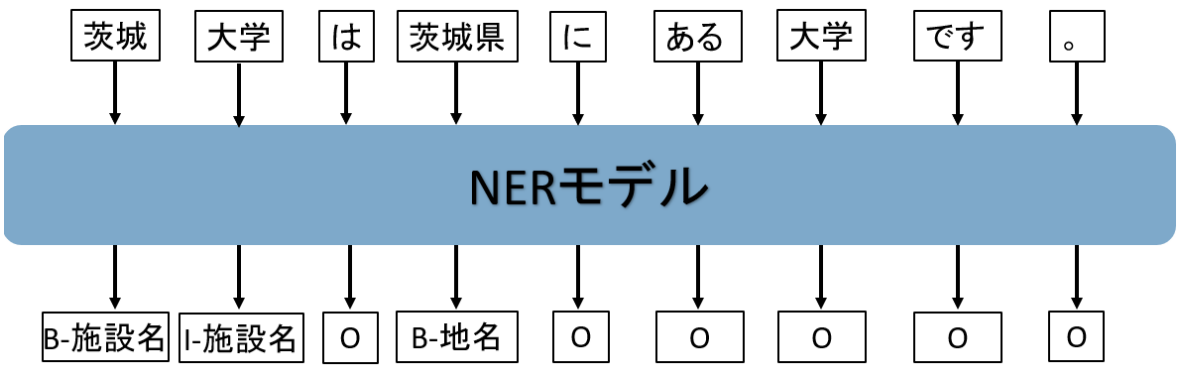

図 1.1: NER の例

Wang らもこのラベルノイズの影響について指摘している [3].Wang らによると NER ベンチマークの 1 つである CoNLL03 データセットにおいて, 約 5.38% のラベル付けミスが確認されている．さらに, これらのラベルノイズを修正したデータを用いることによって, 性能向上が確認されている．Wang らは同論文内で, ラベルミスを処理するための汎用的なフレームワークとして CrossWeigh を提案している．このフレームワークを利用することで, データ内のそれぞれの文章に対して再重み付けが行われ, 生成された重み付けデータを利用することによってラベルノイズの影響を軽減することができる．このような, より正確なデータを使って学習することでモデルの性能向上を図るアプローチは Data-Centric AI(DCAI) と呼ばれ, 注目を集めている [4] [5].

本研究では、以下の二つの実験を行った。

1. CrossWeigh を日本語の固有表現認識に適用させる実験
2. CrossWeigh によって生成される重み付きデータを調査することによってデータセット内にある誤りの発見を試みる実験

1. の実験では CrossWeigh をストックマーク株式会社が提供している, Wikipedia の日本語 NER データセット [6] に適用し, 重み付けデータの生成を行う．さらに, この重み付きデータを用いて, NER モデルの Flair [7] を学習することによって, 日本語での CrossWeigh の効果を確認する．

2. の実験では, CrossWeigh によって生成された重み付きデータの中で、最も低い重みが付けられたデータ全てに対して, 発見したラベルノイズと NER モデルの予測スコアとの関連度に応じてタイプ分類を行い, それら 2 つの相関性について調査を行った．それに加えて、発見したラベルノイズを修正したデータと削除したデータを学習データとしてモデルを構築することで、モデルの性能に変化があるかを実験した。

また, 先行研究 [8] では, データセット内のラベルノイズの調査を, CrossWeigh によって生成された重み付きデータの中で最も低い重みが付けられたデータから, 無作為に抽出した 100 件のデータのみ調査を行った.

さらに、その後の先行研究 [9] では, 最も低い重みが付けられたデータ全てに対して調査を行った. 加えて, 発見したラベルノイズと NER モデルの予測スコアとの関連度に応じてタイプ分類を行い, それら 2 つの相関性についても調査を行った.

本研究では、追加実験として先行研究 [9] において発見したラベルノイズを修正した学

習データ、削除した学習データを作成し、それらのデータを使って学習した NER モデルにどのような変化が起きるのかを実験した。

第 2 章

関連研究

2.1 BERT

自然言語処理 (Natural Language Processing; NLP) の機械学習において、どのようにして対象となる文章をコンピュータが理解できる形に表現するかがおおこな問題となる。文章中の情報をうまく表現することができれば、その後の分類や系列ラベリングなどの様々なタスクを実現することができる。

これらの問題は、近年ではディープラーニングによってより良い表現を獲得することによって、大きな成果を挙げている。その中でも BERT [1] は様々な自然言語処理タスクで、その当時の最先端のモデルの性能を大きく上回る数値を出した。

BERT は Bidirectional Encoder Representations from Transformers の略称で、2018 年に Google が発表したニューラル言語モデルである。BERT は事前学習を使った言語の表現学習ができる。その結果、文脈を深く考慮した表現を獲得することができ、多くの言語タスクで高い性能を発揮した。

2.1.1 BERT の学習

BERT の学習は、はじめに、利用するコーパス内の連続する文を、SEP トークンと呼ばれる文の切れ目を表すトークンを用いてつなげ、長い 1 つの文とした形で入力する。次に、その文からランダムに 1 部のトークンを MASK トークンで置き換える。そして、その MASK トークンに置き換えられた単語を予測する、というタスクが解けるように学習を進めていく。このタスクは Masked Language Modeling と呼ばれ、大量の注釈無し

テキストデータを使って解くことができるため、より多くのデータを使って学習を進めることができる。

Masked Language Modeling によって、BERT はテキストを理解できる事前学習済みモデルを構築することができる。この事前学習済みモデルを言語タスクに利用する時は、最後の MASK トークンを予測する部分を取り除いたネットワークに、解きたいタスク特有のネットワークを付け足して、教師ありデータを用いてファインチューニングすることで様々なタスクに応用することができる。

2.1.2　BERT の構造

BERT は、図 2.1 のような構造をしている。モデルとして Transformer というモデルで提案された Transformer Encorder と呼ばれる、自己注意機構を用いたニューラルネットワークを用いている。自己注意機構を利用することで、文中の周囲の単語から情報を自由に集めることができるだけでなく、離れた位置にある単語の情報も適切に取り入れることができるため、より文脈を深く理解した分散表現を生成することができる。また、入力されたトークンに対する出力は、独立に計算することができるため、並列化による高い計算効率が期待される。

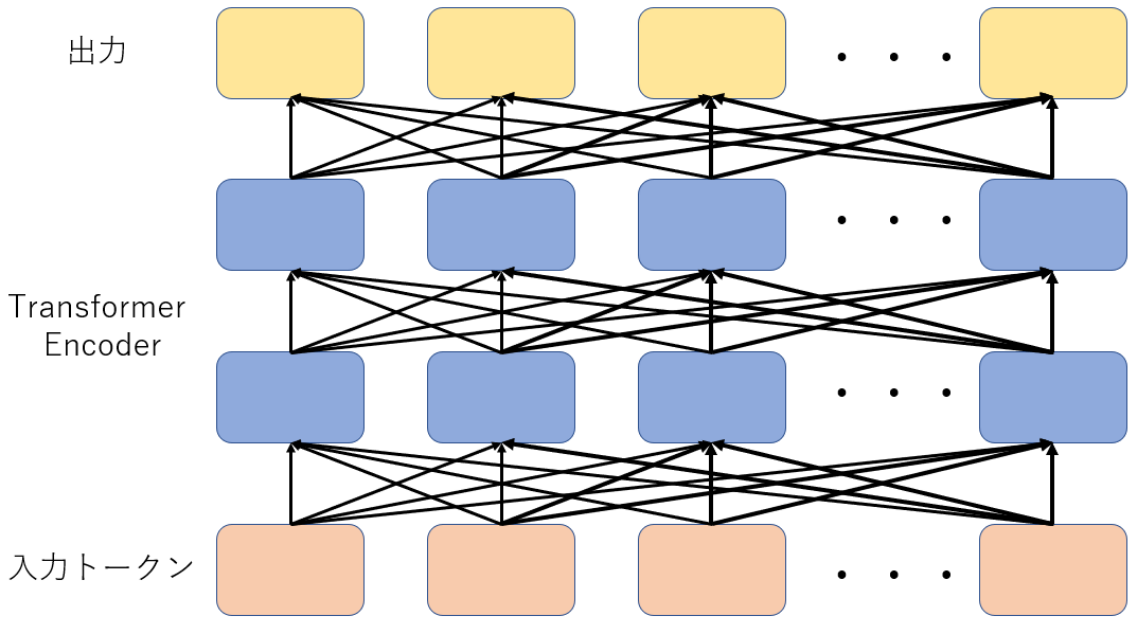

図 2.1: BERT の構造

2.2 Data-Cetnric AI

Data-Centric AI(DCAI) とは、「A Chat with Andrew」というオンラインイベントの中で、DeepLearning.AI 社の Andrew Ng 氏が「MLOps: From Model-centric to Data-centric AI」[*1]といスピーチ内で発表したアプローチ方法の 1 種である。

従来の機械学習では、モデルの性能を向上させるためには、モデルの中で使われるアルゴリズムを改良することによって、より良いモデルを構築するというアプローチ方法が主流であった。Data-Centric AI の考え方では、モデルの中で使われるアルゴリズムではなく、モデルの学習時に利用するデータに焦点を当てて、モデルの性能向上を図るというアプローチ方法である。

実際に「MLOps: From Model-centric to Data-centric AI」のスピーチ内では、500 件のデータの中に 12% のラベルノイズが含まれていた場合、それらのラベルノイズを修正することで、1000 件のデータを用いて学習したモデル以上の性能を発揮したという結果が報告されている。このように、より質の高いデータを利用することで、同じ機械学習モデルでも性能向上が期待できる。

「Data-Centric AI Competition」[*2]という使用するモデルを共通にして、使用するデータセットに対して改良を加えて、モデルの性能を競うコンペティションが開催されていて、様々な手法が提案され、注目を集めている。

2.3 固有表現認識

地名や人名、会社といった固有名詞に、数量や日付、時間などを加えたものを固有表現 (Named Entity; NE) と呼ぶ。そして、与えられたテキストからそれらの固有表現を見つけ出すタスクを固有表現認識 (Named Entity Recognition; NER) という。

固有表現認識は入力テキスト中の特定の情報を抜き出すということから、情報抽出 (Information Extraction) の 1 つとされている。

また、入力テキスト中の単語の列に対して、ラベルの列を推定する系列ラベリングと呼ばれるタスクである。

[*1] https://www.youtube.com/watch?v=06-AZXmwHjo
[*2] https://https-deeplearning-ai.github.io/data-centric-comp/

2.3.1　関根の拡張固有表現階層

関根の拡張固有表現階層 [10] とは、固有表現の定義の一つである。

一般的な内容を対象にした質問応答システム、幅広い分野を対象とした情報抽出、機械翻訳、情報検索や要約などの幅広い自然言語処理のタスクに対応することを目的として考案され、組織名という固有表現に対して、政治的組織名や法人名、競技組織名などの下位の固有表現を階層的に定義している。

2.3.2　IOB2 フォーマット

IOB2 フォーマットとは、固有表現認識を系列ラベリングのタスクとして解く際に用いられる、ラベル列の形式のことである。BIO 形式とも呼ばれ、Beginning, Inside, Outside のそれぞれの頭文字を取っている。

固有表現となる単語は、モデルに入力するために行われるトークナイズの際に、1 つの固有表現であっても、複数のトークンによって構成されることが多い。それらのトークンに適切なラベリングを行うために、IOB2 フォーマットは用いられる。

具体的には、固有表現の 1 つのトークンで構成される固有表現や、複数のトークンによって構成される固有表現の先頭となるトークンに対して、「B-(固有表現名)」ラベルを付与する。固有表現の先頭でないトークンに対しては「I-(固有表現名)」のラベルを付与する。それ以外の、固有表現でないトークンに対しては「O」ラベルが付与される。

NER モデルでは、入力されたトークンごとに分割されたテキストに対して、これらの BIO タグを適切にラベリングすることを目的としている。

2.3.3　F 値

固有表現認識のタスクでは、評価指標として F 値 (F1-score) がよく用いられる。F 値は再現率 (Recall) と適合率 (Precision) の調和平均によって計算される。

$$F1\text{-}score = \frac{2Precision * Recall}{Precision + Recall} \tag{2.1}$$

2.4　spaCy

自然言語によって書かれた文章をニューラルネットワークに入力しようとする場合、まず文の構成要素であるトークンごとに分割して、各トークンを分散表現と呼ばれるベクトルにするという処理を行うのが一般的である。この処理によって、私たちが使用する文章をニューラルネットワークに入力してモデルを構築し、様々なタスクを解くということができる。

英語のように単語の間のスペースによって、はじめから文章が単語ごとに分割されている言語に関しては、ほとんどの場合において、文を単語に区切るという処理を必要としない。しかし、日本語のように単語間にスペースがない言語は、言葉が意味を持つまとまりの単語の最小単位である形態素に分割する必要がある。

spaCy[*3]は高度な自然言語処理を行うためのライブラリであり、訓練済みのモデルを読み込むことによって、上記の日本語における文を形態素に分割する処理が可能になる。日本語の訓練済みモデルとしてはリクルートと国立国語研究所の共同研究成果であるGiNZA[*4]が利用できる。

2.5　ラベルノイズの検知

CrossWeigh のように, ラベルの誤りを自動的に検出する試みは以前から研究されている [11] が, この研究では品詞の間違いに対して修正を行うため, 今回のように NER に対して適用することができない. 他にも [12] [13] [14] ではラベルノイズが含まれる学習データを利用した NER の研究がされているが, これらでは NE が O ラベルとなる誤りに対してのみを対象としている.

日本語データセットにおけるラベルの誤り問題に取り組んだ研究として [15] が挙げられる. この研究では, アノテーション漏れを推定しそのエンティティを学習データに追加することによって, 系列ラベリングを用いたエンティティ抽出の再現率の向上を行っている. また, [16] では Teacher-Student 学習を利用することでラベル誤りを含むデータにおける NER の性能向上について研究を行い, CrossWeigh と同様にラベルノイズの影響を

[*3] https://spacy.io/
[*4] https://megagonlabs.github.io/ginza/

緩和し性能向上が確認されている.

第 3 章

CrossWeigh

CrossWeigh とは Wang らが提案した、ラベルノイズを汎用的に処理するためのフレームワークである [3]。CrossWeigh は以下の 2 つのモジュールから構成されている.

1. ラベルノイズの推定を行うモジュール
2. 推定したラベルノイズが含まれているデータに再重み付けを行うモジュール

以降の節ではこの 2 つのモジュールごとに解説を行う。

3.1 ラベルノイズの推定

1 つ目のラベルノイズの推定は k-分割交差検証をもとにした考えで実装されている. k-分割交差検証とは、使用するデータセットを k 個に分割し、分割したデータの内 1 つをテストデータ、それ以外のデータを学習データとして利用してモデルを構築するという操作を、分割したデータそれぞれ 1 回ずつテストデータになるように行うというものである。さらに、それぞれの操作で構築した k 個のモデルの精度を平均化することで、そのモデル全体での精度を評価するという方法である。k=5 としたときの例を図 3.1 に示す。

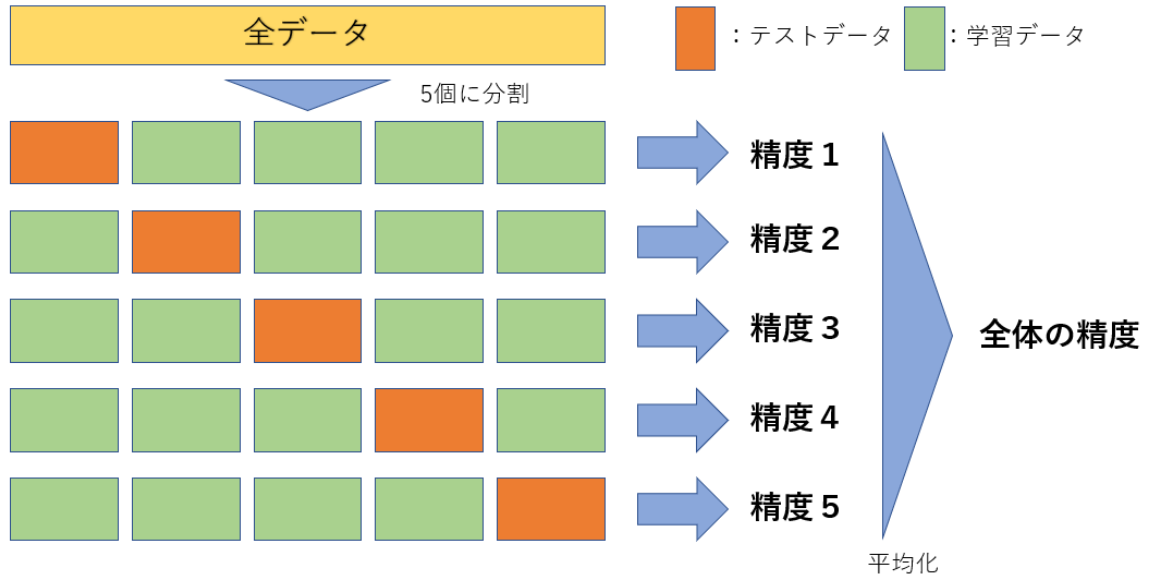

図 3.1: k-分割交差検証 (k=5 の場合)

CrossWeigh では、まず学習に使用するデータを k 個に分割することで $D_1, D_2, \ldots, D_k$ を作成する．そして, NER モデル $M_i(1 \leq i \leq k)$ を学習データの中から D_i を除いたデータによって学習させることで k 個のモデルを構築する．k=10, i=4 とした場合の例を図 3.2 に示す。

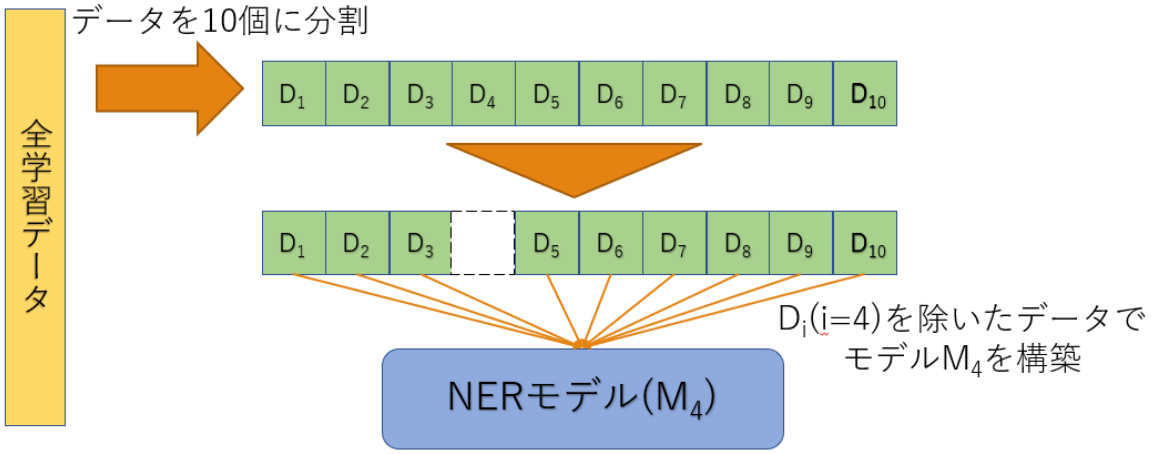

図 3.2: k 個のモデルを作成 (k=10, i=4 の場合)

上記の操作によって構築した各モデル M_i を使って D_i の文のラベルに対して予測させ, モデルの出力と異なるラベルを持つ文は, ラベルノイズである可能性が高いものとしてマークする.

図 3.3 では、「茨城大学」という固有表現に対して、データセット内では「茨城」というトークン部分のみに地名のラベルが付与されているが、モデルの予測では「茨城大学」というトークンに施設名のラベルが付与されるとしている。この結果から、「茨城大学は茨城県にある大学です。」という文章はラベルノイズが含まれている可能性が高いとしてマークがされる。

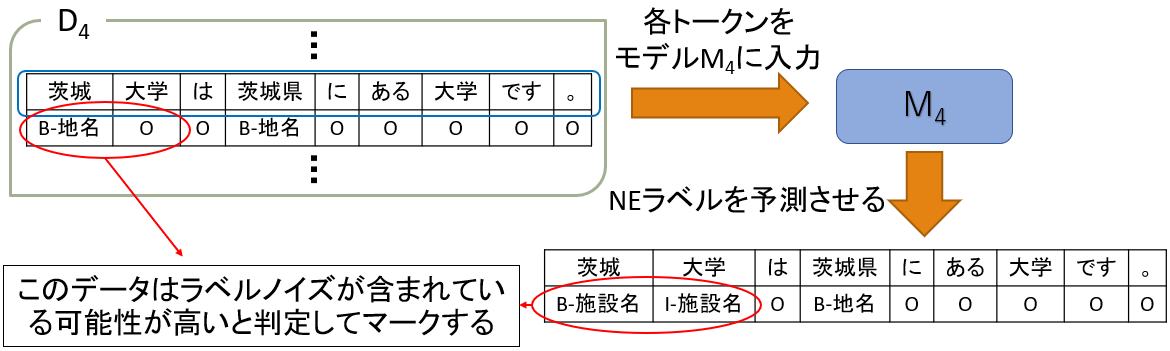

図 3.3: ラベルノイズの推定

以上の 2 つの処理を, それぞれ異なるランダムな分割で行い, t 回実行する. ここまでを実行することで, データセットのラベルノイズの推定が行われる. 本実験では CrossWeigh の元論文 [3] で設定されていた $k = 10, t = 3$ を参考にして行った.

3.2　再重み付け

2 つ目のモジュールである再重み付けは, ラベルノイズの推定でマークされた文 x_i に対して重み w_i を調整することで実装されている. 最初に, 全ての文の重みを $w_i = 1$ として設定する. 次に, ラベルノイズとしてマークされた文の重みを**式** (3.1) で計算する.

$$w_i = \epsilon^{c_i} \tag{3.1}$$

式 (3.1) の ϵ はパラメータであり, ラベルノイズ推定モジュールの精度に応じて設定する. 本実験では $\epsilon = 0.7$ とした. c_i は t 回実行されたラベルノイズの推定で, 何回モ

デルの出力と異なるラベルが推定されたかによって決まる．つまり c_i の取りうる値は $1 \leq c_i \leq t$ のいずれかの整数値である．

CrossWeigh ではこれら 2 つのモジュールを使って, ラベルノイズの影響を緩和する重み付きデータを作成する．

第 4 章

データセット内のラベルノイズの検出

CrossWeigh を適用することで重み付きデータが作成される．この重み付きデータは各文ごとにラベルミスに応じた重みがつけられているため, この重みが最小のものがラベルノイズが含まれている可能性が高いデータであると考え, 検出を試みた．ラベルノイズであるか否かの判定は, Wikipedia の日本語 NER データセットを構築する際に使われた関根の拡張固有表現階層 [10] を参考にした．

また、ラベルノイズが含まれたデータは NER モデルの分類スコアが低くなるのではないかと考え、上記の実験で発見したラベルノイズを含む文章を再度 NER モデルに入力し、予測結果の分類スコアを見ることで、ラベルノイズと分類スコアの相関性についても調査を行った。

第 5 章

実験

5.1 CrossWeigh の日本語適応

5.1.1 使用データセット

本研究では, データセットとして, ストックマーク株式会社が提供している, Wikipedia の日本語 NER データセットを用いた. このデータセットは 5.1 のような json 形式で配布されている。

ソースコード 5.1: ner.json

```
{
      "curid": "2415078",
      "text": レッドフォックス株式会社は、東京都千代田区に本社を置く
          "サービス企業である。IT",
      "entities": [
          {
              "name": レッドフォックス株式会社"",
              "span": [
                  0,
                  12
              ],
              "type": 法人名""
          },
          {
              "name": 東京都千代田区"",
              "span": [
                  14,
                  21
```

```
                ],
                "type": 地名""
            }
        ]
    }
```

CrossWeigh および固有表現認識のタスクに適用するために, spaCy と GiNZA を用いて表 5.1 のように整形を行った. 固有表現認識のラベルのフォーマットとしては IOB2 フォーマットを利用した. また, 整形したデータを学習データ, 検証用データ, テストデータの比率が 8:1:1 になるように分割した.

表 5.1: データ例

トークン	ラベル
レッドフォックス	B-法人名
株式会社	I-法人名
は	O
,	O
東京都千代田区	B-地名
に	O
本社	O
を	O
置く	O
IT	O
サービス	O
企業	O
で	O
ある	O
.	O

5.1.2　使用モデル

NER モデルは, Flair を用いた. ここで用いる Flair とは Akbik らの Contextual String Embeddings for Sequence Labeling [7] で提案された NER モデルを, 日本語データを用いて事前学習を行ったものである. Contextual String Embeddings for Sequence Labeling とは文字ベースの言語モデルを利用することで単語分散表現を構築し、その分散表現を利用することで単語ごとの固有表現ラベルを予測するというモデルである。このモデルのアーキテクチャを図 5.1 に示す。

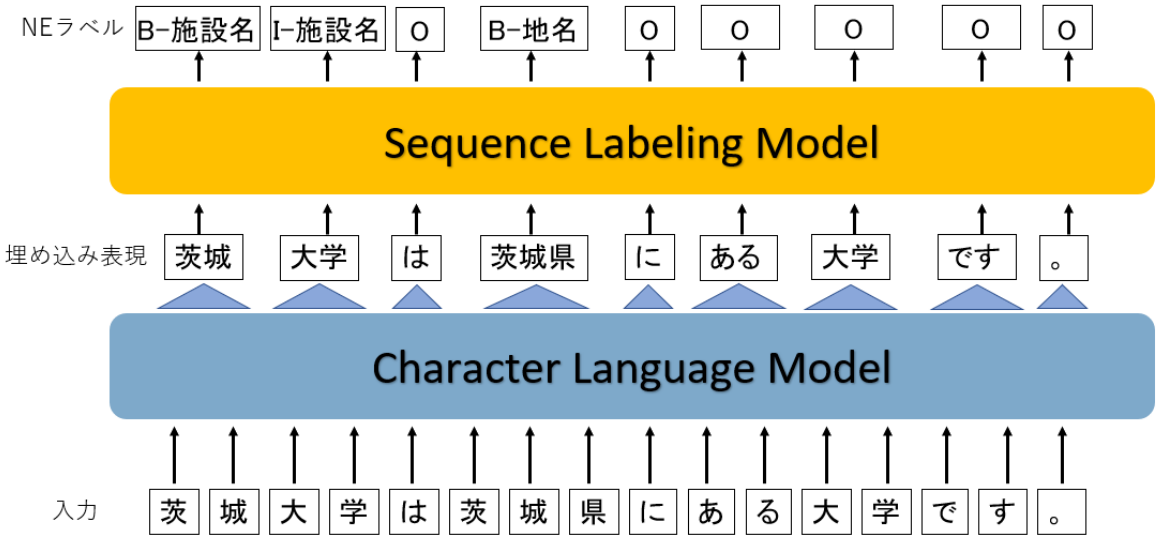

図 5.1: Contextual String Embeddings for Sequence Labeling のアーキテクチャ

さらに, CrossWeigh によって付与された重みを考慮して学習できるようにモデルを調整した.

5.1.3　評価方法

評価指標として F1-score(F 値) を用いた. CrossWeigh を適用していないデータ (Original data) と CrossWeigh を適用したデータ (Weighed data) それぞれに対して学習を行い, モデルがトークン列に対して適切なラベルを付与できたか否かをもとに算出を行った.

5.1.4　結果

Flair の結果を表 5.2 に示す．F1-score に関しては micro, macro のどちらにおいても CrossWeigh を適用したものが良い性能を示している．それぞれのラベルに対する結果では, 最大で 0.1 ポイント程のスコア向上がみられた．これらのことから日本語においても CrossWeigh の有効性がわかる．しかし, ラベルによってはポイントが減少しているものもあるため, これらに対しては今後の課題として分析が必要であると考えられる．

表 5.2: Flair の NER モデルにおけるラベルごとの結果

	Base line	Original data	Weighed data
その他の組織名	0.81	0.7674	0.8625
イベント名	0.84	0.8367	0.8324
人名	0.95	0.9341	0.9181
地名	0.87	0.9041	0.8932
政治的組織名	0.8	0.7863	0.8889
施設名	0.81	0.7800	0.8545
法人名	0.88	0.8755	0.8756
製品名	0.73	0.8117	0.7810
F1(micro)	0.86	0.8640	0.8758
F1(macro)		0.8370	0.8633

5.2　ラベルノイズの検出

5.2.1　ラベルノイズ

今回使用した Wikipedia の日本語 NER データセットにおけるラベルノイズの例として**図** 5.2 が挙げられる．(a) では文中の「旭川」と「小樽」に対して地名のラベルが付与されるべきであるが, ラベルが付与されていないため O タグとして処理が行われてしまう．また, (b) では「水夏希は」に人名のラベルが付与されてしまっているが, スパンが正しくないために「は」も I-人名タグとして処理が行われてしまう．これらのラベルノイズは CrossWeigh の再重み付けの際に, より小さい重みが付与されていると考えられる．

トークン	旭川	の	「	北海日日新聞	」	を
ラベル	O	O	O	法人名	O	O

(a) 異なるラベルが付与されているデータ

トークン	水夏希は	、	日本	の	女優	。
ラベル	人名	O	地名	O	O	O

(b) スパンがずれているデータ

図 5.2: ラベルノイズ例

5.2.2　結果

CrossWeigh によって生成された重み付きデータを調査したところ, データセットの全 5343 個のデータの内, 1484 個のデータに重みが最小のものが付与されていることがわかった．これらの 1484 個のデータを, トークンごとのラベルの分類スコアに着目して調査した結果, 大きく分けて以下のような 4 つのタイプに分類できた．

1. 分類スコアが最も低いトークンがラベルノイズであるもの
2. モデルの予測結果が間違っており, 元のデータがあっているもの
3. 分類スコアが最も低いトークン以外のものがラベルノイズであるもの
4. 予測結果と元のデータどちらもあっているもの

以上 4 つのタイプは以下の表 5.3 にまとめたデータ数であることがわかった.

表 5.3: タイプ分類の結果

Type	データ数
1	16
2	416
3	57
4	995

このタイプ分類からラベルノイズとなるものは Type1, 3 となる. つまり, 最小の重みが付与された 1484 個のデータの内, 73 個のデータがラベル付けに誤りがあるデータであることがわかった.

第 6 章

考察

6.1 CrossWeigh の日本語 NE データへの適用

Flair の結果である表 5.2 から, Original data で学習した Flair よりも, Weighed data で学習した Flair の方が NE の識別精度が向上していることがわかる. このことから CrossWeigh の手法は日本語 NE データにおいても有効であることがわかった.

しかし, この結果をグラフ化した**図** 6.1 に示したラベルごとの結果を見ると, 全てのラベルに対して識別精度が向上しているのではなく, 大きく精度が向上しているものから減少しているものもあるため, これらを分析, 改善することによってより良い重み付きデータを生成できると考えられる.

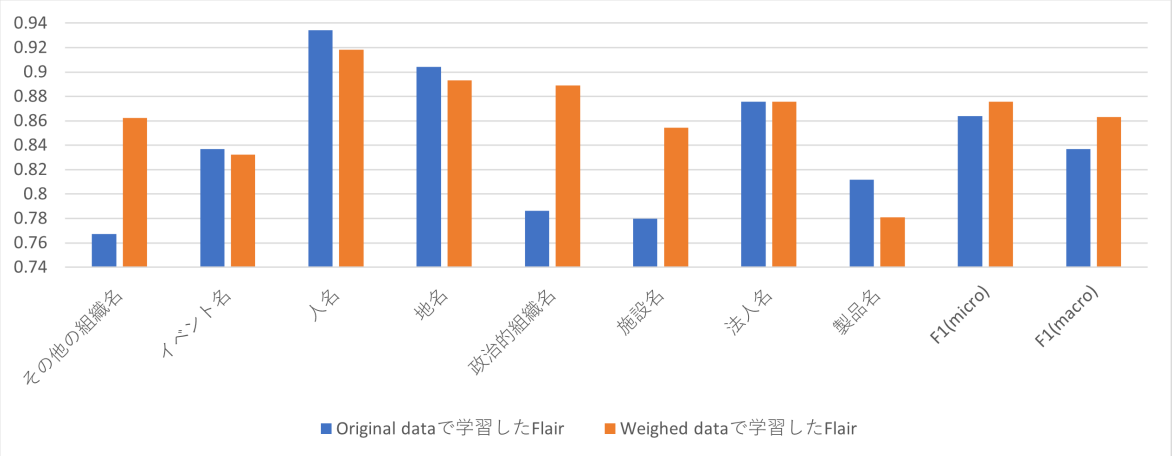

図 6.1: ラベルごとの F 値

6.2　ラベルノイズの特定

5.2 節より, 使用した Wikipedia の日本語 NER データセット内のラベルノイズの探索範囲を狭めることに成功した. CrossWeigh の重み付きデータを利用することで効率的にデータセット内のラベルノイズを発見することができることがわかった.

ラベルノイズと予測の際の分類スコアの関係としては, あまり大きな相関は見られなかった. これには, 予測に使うモデルがラベルノイズを含むデータセットを学習に使用してしまっているため, ラベルノイズとなるトークンに対して実際に正しいラベルを予測させるのが難しいためと考えられる.

今回の実験で発見したラベルノイズが, 元のデータではどのラベルが付けられていて, 実際にはどのラベルが付くべきであったかを付録の表 6.1 にまとめた. ここから, 調査した Wikipedia の日本語 NER データセット内のラベルノイズでは, 元のラベルが O ラベルであるものが最も多いことがわかった. ここからラベルノイズの発生原因としては, 固有表現となる単語を見落としてしまうというものが最も多いことがわかった. CrossWeigh の有効性から考えられるように, ラベルノイズがモデルに与える影響は大きいため, どのようにしてデータセットのラベルノイズの発生を抑えることが今後の課題であることがわかった.

表 6.1: ラベルごとの修正内訳

修正後のラベル ＼ 修正前のラベル	その他の組織名	イベント名	人名	地名	政治的組織名	施設名	法人名	製品名	O
その他の組織名		0	1	0	2	0	3	0	3
イベント名	0		0	0	0	0	0	1	3
人名	0	0		1	0	0	6	1	6
地名	0	0	0		0	2	0	0	11
政治的組織名	0	0	2	1		1	6	0	0
施設名	0	0	0	2	0		0	1	1
法人名	1	0	0	1	0	0		0	1
製品名	0	1	0	1	0	0	1		12
O	0	1	2	0	0	0	1	2	

6.3　追加実験

6.3.1　実験の目的

5.2 節での CrossWeigh によって生成された重み付きデータを用いることによって、モデルの学習に利用したデータの中から 73 個のラベルノイズを発見することができた。本節では、これらの発見したラベルノイズに対して、ラベルノイズとなる部分を修正したデータと、ラベルノイズが含まれている文を削除したデータのそれぞれを学習データとしてモデルを構築すると、モデルの固有表現ラベルに対する識別精度において、どのような変化が起きるのかを確認するために追加実験を行う。

6.3.2　実験設定

まずは、使用する学習データの作成を行った。A.1 節と同様にデータの整形を行い、学習データ、テストデータ、検証用データの内訳が同じになるようにした。その中の学習データに対して、ラベルノイズの修正を行ったデータ (Corrected data) とラベルノイズが含まれた文を削除したデータ (Deleted data) の 2 つを作成した。

また、NER モデルとしては 5.1 で使用した Contextual String Embeddings for Sequence Labeling で提案されたモデル (Flair) に加えて、より出現頻度の低い単語に対して良い分散表現を得ることができる Pooled Contextualized Embeddings for Named Entity Recognition [17] で提案されたモデル (Pooled Flair) の 2 つをそれぞれ構築し、F1-score(F 値) を評価指標として識別精度の確認を行った。

6.3.3　実験結果

実験の結果を、それぞれのモデルである Flair と Pooled Flair ごとに表 6.2, 6.3 に示す。比較用として Original data と Weighed data を載せる。

表 6.2: Flair の結果

	Original data	Weighed data	Corrected data	Deleted data
その他の組織名	0.7674	0.8625	0.7791	0.7399
イベント名	0.8367	0.8324	0.8000	0.8342
人名	0.9341	0.9181	0.9306	0.9349
地名	0.9041	0.8932	0.8944	0.8945
政治的組織名	0.7863	0.8889	0.7692	0.7710
施設名	0.7800	0.8545	0.8060	0.8060
法人名	0.8755	0.8756	0.8812	0.8686
製品名	0.8117	0.7810	0.8067	0.7699
F1(micro)	0.8640	0.8758	0.8604	0.8559
F1(macro)	0.8370	0.8633	0.8334	0.8274

表 6.3: Pooled Flair の結果

	Original data	Weighed data	Corrected data	Deleted data
その他の組織名	0.7719	0.8060	0.7746	0.7746
イベント名	0.8205	0.8469	0.8384	0.8265
人名	0.9278	0.9014	0.9335	0.9304
地名	0.8879	0.8177	0.9045	0.9029
政治的組織名	0.7955	0.8325	0.7812	0.7923
施設名	0.8205	0.7826	0.8259	0.7897
法人名	0.8700	0.8058	0.8791	0.8730
製品名	0.7934	0.7573	0.7801	0.8051
F1(micro)	0.8601	0.8283	0.8653	0.8630
F1(macro)	0.8359	0.8188	0.8397	0.8368

表 6.2 より、NER モデルに Flair を用いた場合、Corrected data, Deleted data のどちらにおいても、全体の精度を表す F1-macro, F1-micro のどちらの値も Original data よりも低い数値となっているため、識別精度が下がっていることがわかる。

表 6.3 より、NER モデルに Pooled Flair を用いた場合は、Corrected data, Deleted data を用いて学習した場合は、Original data, Weighed data のどちらにおいても F1-macro, F1-micro の値が高くなっていることから、Corrected data, Deleted data は Pooled Flair を用いた際に識別精度が向上することがわかった。

また、どちらの NER モデルを用いた場合でも、Corrected data と Deleted data を比較した場合は、Corrected data の方が識別精度が高いということがわかった。

6.3.4　考察

表 6.2 の結果をグラフにしたものが図 6.2 である。ラベルごとの F 値を見ると、Corrected data では、イベント名のラベルに対する値が Original data よりも大きく低下していることがわかる。

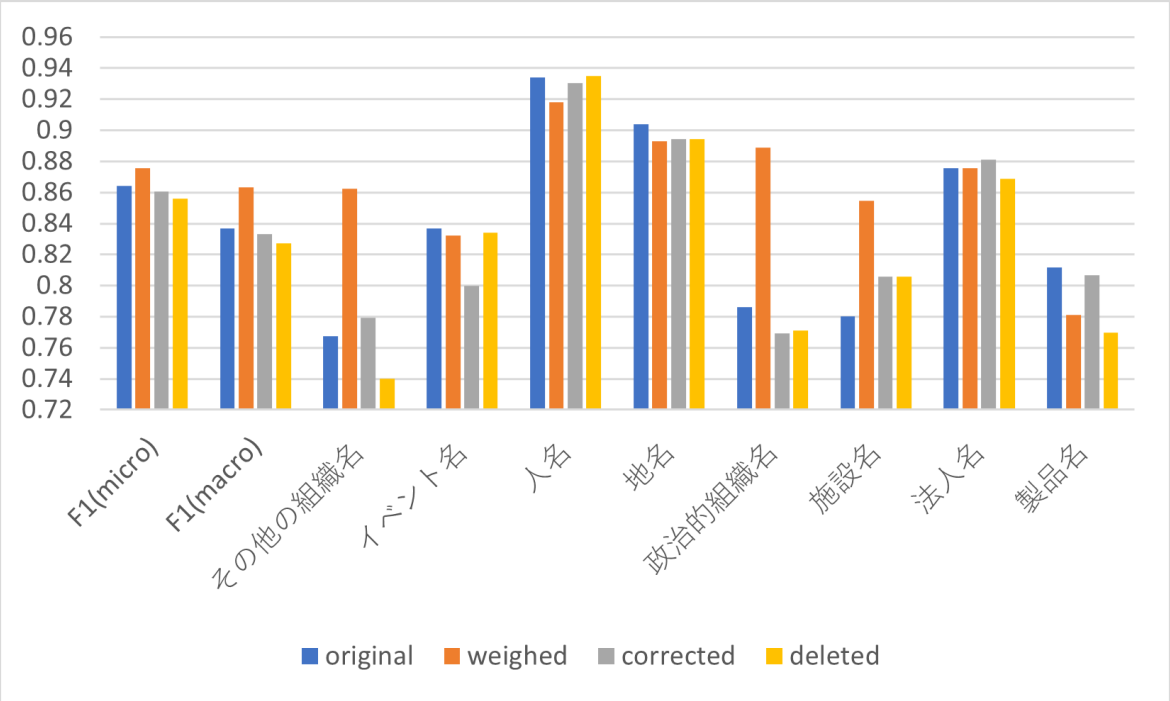

図 6.2: Flair でのラベルごとの F 値

このことに着目し、イベント名のラベル予測に対して、Original data を用いて学習した NER モデルではラベルの予測があっていたが、Corrected data を用いて学習した NER モデルでは予測が間違っていた単語について調査した結果、以下に示す 7 単語が見つかった。

1. 第 1 回北伐
2. クリスマス・メッセージ
3. 第 10 回大会
4. 讃岐うどんブーム
5. 1972 年アイランド・ツアー
6. 草レース
7. 国際プロレスアワー

上記の 7 単語の中でも、2, 3, 6 の単語は、使用したデータセット内では、O ラベルが正解となっていた。Original data を用いて学習した NER モデルでの予測では、O ラベルで正解となっていたが、Corrected data を用いて学習した NER モデルでは、イベント名のラベルを付与してしまったために不正解となっていた。これらの単語は文章の中での使われ方によっては、イベント名ともいえるような曖昧性の高い単語であり、正しい予測が困難であったと考えられる。

また、4, 7 の単語においては、4 の単語では地名のラベルが、3 の単語ではその他の組織名のラベルが使用したデータセット内での正解となっている。どちらの単語もそれの正解ラベルに加えて、イベント名のラベルも付与されるような 2 個以上のラベルが付く単語であると考えられる。しかし、NER モデルの識別器の性質上、2 つ以上のラベルを 1 つの単語で予測することは不可能なため、この問題は固有表現認識というタスクにおける大きな問題であると考えられる。

表 6.3 の結果をグラフにしたものが図 6.2 である。ラベルごとの F 値を見ると、Corrected data では、政治的組織名のラベルに対する値が Original data よりも大きく低下していることがわかる。

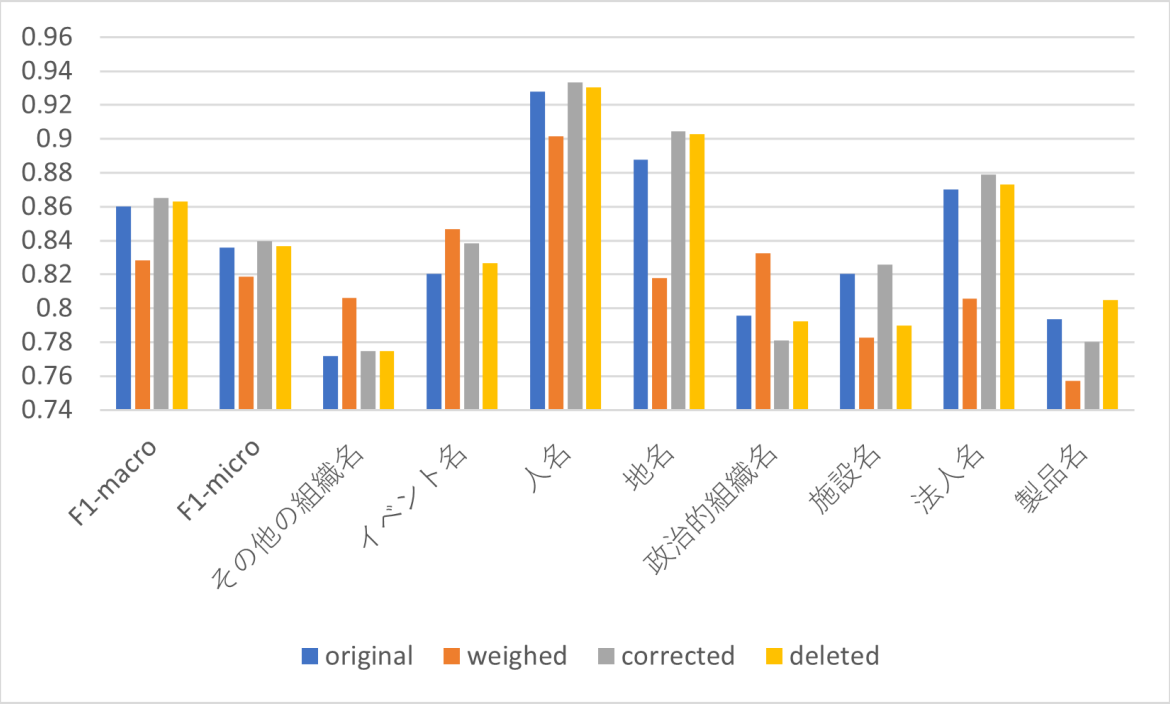

図 6.3: Pooled Flair でのラベルごとの F 値

同様にして、政治的組織名のラベル予測に対して、Original data を用いて学習した NER モデルではラベルの予測があっていたが、Corrected data を用いて学習した NER モデルでは予測が間違っていた単語について調査した結果、以下に示す 10 単語が見つかった。

1. 李万建党中央委員会
2. 国連
3. FBI
4. 沿岸警備隊
5. 海軍
6. 評議会
7. レーテ共和国
8. 東ローマ帝国
9. 太平天国
10. 文部科学大臣

上記の単語では、4, 5, 6, の単語が、使用したデータセット内では O ラベルが正解となっていた。Original data を用いて学習した NER モデルでの予測では、O ラベルで正解と

なっていたが、Corrected data を用いて学習した NER モデルでは、政治的組織名のラベルを付与してしまったために不正解となっていた。これらの単語は、先述の部分と同様に文章の中での使われ方によっては、政治的組織名ともいえるような曖昧性の高い単語であり、正しい予測が困難であったと考えられる。

また、1, 7, 8, 9, の単語では、政治的組織名のラベルに加えて、1 の単語では人名のラベルが、7, 8, 9 の単語では地名のラベルが付与されるような単語であると考えられるため、正しい予測が困難であったと考えられる。

第 7 章

結論

本研究では日本語 NER データセットにおける CrossWeigh の有効性について検証を行った. CrossWeigh によって生成された重み付きデータを利用することで NER モデルの性能向上を確認することができた. 今後の課題は精度が下がったラベルに対しての分析である.

また, データセット内のラベルノイズの調査に関しては, CrossWeigh が生成する重み付きデータを利用することで, 実際にデータセット内のラベルノイズを発見することに成功した. ラベルノイズとなるトークンはモデルの予測結果がよい数値ではないと考え, 2 つの関係について調べてみたが, 相関性は見られなかった. ラベルノイズの内訳についても調査したが, 最も多いラベルノイズの発生原因はラベルの付け忘れであることがわかり, データセットを構築する際に, このようなミスの発生を抑制することが今後の課題である.

追加実験においては、ラベルノイズとなる部分を修正したデータと削除したデータを学習に用いることの有効性について調査した。その結果、NER モデルによってラベルの識別精度が向上するものや、減少してしまうものがあるということがわかった。識別精度が減少してしまう要因としては、人間にとっても正解となるラベルが判断しにくい単語や、1 つの単語に 2 つ以上のラベルが付与される単語が存在しているということが考えられる。こういった単語に対して NER モデルがうまく予測できるようにするのか、それとも固有表現認識というタスクに対して見直しが必要であるのかは、今後の課題である。

謝辞

本研究を進めるにあたり, 多くのご指導を頂いた指導教官の新納浩幸教授に感謝申し上げます. また, 研究室での活動を通じて, 多くの知識や示唆を頂いた新納研究室の皆さまに感謝致します.

参考文献

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization, 2016.

[3] Zihan Wang, Jingbo Shang, Liyuan Liu, Lihao Lu, Jiacheng Liu, and Jiawei Han. CrossWeigh: Training named entity tagger from imperfect annotations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 5154–5163, Hong Kong, China, November 2019. Association for Computational Linguistics.

[4] Mark Mazumder, Colby Banbury, Xiaozhe Yao, Bojan Karlaš, William Gaviria Rojas, Sudnya Diamos, Greg Diamos, Lynn He, Douwe Kiela, David Jurado, David Kanter, Rafael Mosquera, Juan Ciro, Lora Aroyo, Bilge Acun, Sabri Eyuboglu, Amirata Ghorbani, Emmett Goodman, Tariq Kane, Christine R. Kirkpatrick, Tzu-Sheng Kuo, Jonas Mueller, Tristan Thrush, Joaquin Vanschoren, Margaret Warren, Adina Williams, Serena Yeung, Newsha Ardalani, Praveen Paritosh, Ce Zhang, James Zou, Carole-Jean Wu, Cody Coleman, Andrew Ng, Peter Mattson, and Vijay Janapa Reddi. Dataperf: Benchmarks for data-centric ai development, 2022.

[5] 古田拓毅. データ中心の視点から捉える深層強化学習. 人工知能, Vol. 37, No. 4, pp. 507–515, 2022.

[6] 近江崇宏. Wikipedia を用いた日本語の固有表現抽出のデータセットの構築. 言語処

理学会第 27 回年次大会発表論文集 (NLP2021), 2021.

[7] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pp. 1638–1649, 2018.

[8] 西村柾人, 新納浩幸. Crossweigh の日本語 ner データセットへの適用. Technical Report 21, 茨城大学工学部情報工学科, 茨城大学大学院理工学研究科情報科学領域, sep 2022.

[9] 西村柾人, 新納浩幸. Crossweigh の日本語 ner データセットへの適用. 言語処理学会第 29 回年次大会発表論文集 (NLP2023) to appear (2023), 2023.

[10] Satoshi Sekine, Kiyoshi Sudo, and Chikashi Nobata. Extended named entity hierarchy. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, Las Palmas, Canary Islands - Spain, May 2002. European Language Resources Association (ELRA).

[11] 中川哲司, 松本雄二. サポートベクターマシンを用いた誤り検出. 言語処理学会年次大会発表論文集, Vol. 8, pp. 563–566, 2002.

[12] Kun Liu, Yao Fu, Chuanqi Tan, Mosha Chen, Ningyu Zhang, Songfang Huang, and Sheng Gao. Noisy-labeled ner with confidence estimation, 2021.

[13] Zhanming Jie, Pengjun Xie, Wei Lu, Ruixue Ding, and Linlin Li. Better modeling of incomplete annotations for named entity recognition. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 729–734, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[14] Stephen Mayhew, Snigdha Chaturvedi, Chen-Tse Tsai, and Dan Roth. Named entity recognition with partially annotated training data, 2019.

[15] 伊藤雅弘, 山崎智弘. アノテーション漏れ推定を用いたエンティティ抽出. 言語処理学会第 27 回年次大会発表論文集 (NLP2021), 2021.

[16] 田川裕輝, 中野騰久, 尾崎良太, 谷口友紀, 大熊智子, 鈴木裕紀, 木戸尚治, 富山憲幸. Teacher-student 学習を利用したラベル誤りを含むデータにおける固有表現認識の性能向上. 言語処理学会第 28 回年次大会発表論文集 (NLP2022), 2022.

[17] Alan Akbik, Tanja Bergmann, and Roland Vollgraf. Pooled contextualized embeddings for named entity recognition. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 724–728, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

付録

A プログラムリスト

本研究で利用した主要なプログラムは以下のとおりである。

A.1 json 形式のデータセットをモデルに入力できる形にするプログラム

A.2 モデルを構築するプログラム

A.3 テキストを BIO 形式に変換するプログラム

ソースコード A.1: create_dataset.py

```
1  import json, random
2  import numpy as np
3  import spacy
4
5  def create_tokens_and_labels(nlp, splitted):
6              tokens = []
7              labels = []
8              for s in splitted:
9                      text = s['text']
10                     label = s['label']
11                     tokens_splitted = nlp(text)
12                     if label == "O":
13                         labels_splitted = ["O"] * len(
                               tokens_splitted)
14                     else:
15                         labels_splitted = [f"B-{label}"]
16                         labels_splitted.extend([f"I-{label}"] * (
                               len(tokens_splitted) - 1))
17                     tokens.extend(tokens_splitted)
18                     labels.extend(labels_splitted)
```

```
19
20              return tokens, labels
21
22      def create_tagged_data(nlp, text, entities):
23              entities = sorted(entities, key=lambda x: x['span'
                    ][0])
24              splitted = []
25              position = 0
26              for entity in entities:
27                      start = entity['span'][0]
28                      end = entity['span'][1]
29                      label = entity['type']
30
31                      splitted.append({'text': text[position:start],
                             'label':"O"})
32
33                      splitted.append({'text': text[start:end], '
                            label':label})
34                      position = end
35              splitted.append({'text': text[position:], 'label':"O"
                    })
36              if splitted[0] == {'text': '', 'label': 'O'}:
37                  splitted.pop(0)
38              tokens, labels = create_tokens_and_labels(nlp, splitted
                    )
39              return {"tokens": tokens, "labels": labels}
40
41
42      def create_dataset(nlp, orig_data, filename):
43
44          with open(filename, "w", newline='\n') as f:
45              f.write("-DOCSTART-␣-X-␣-X-␣O\n\n")
46              for data in orig_data:
47                  tagged_data = create_tagged_data(nlp, data["text"],
                        data["entities"])
48                  tokens = tagged_data["tokens"]
49                  labels = tagged_data["labels"]
50                  for i in range(len(tokens)):
51                      f.write(f"{tokens[i]}\t{labels[i]}\n")
52                  f.write("\n")
53
```

```
54
55      nlp = spacy.load('ja_ginza')
56
57      with open("./ner.json", "r") as f:
58          jsons = json.load(f)
59
60      jsons = random.sample(jsons, len(jsons))
61
62      ds = np.array(jsons)
63      indices = [int(ds.size * n) for n in [0.8, 0.8 + 0.1]]
64      train, test, dev = np.split(ds, indices)
65
66      dataset = train.tolist()
67
68      create_dataset(nlp, train.tolist(), "train.txt")
69      create_dataset(nlp, test.tolist(), "test.txt")
70      create_dataset(nlp, dev.tolist(), "dev.txt")
```

ソースコード A.2: model.py

```
1       from flair.data import Corpus
2       from flair.data_fetcher import NLPTaskDataFetcher
3       from flair.embeddings import TokenEmbeddings, WordEmbeddings,
            StackedEmbeddings, FlairEmbeddings, PooledFlairEmbeddings
4       from flair.trainers import ModelTrainer
5       from flair.models import SequenceTagger
6       from typing import List
7
8       column_format = {0: 'text', 1: 'ner'}
9
10      data_folder = "data_correct"
11      model_folder = 'model'
12
13      corpus: Corpus = NLPTaskDataFetcher.load_column_corpus(data_folder,
14                                              column_format=column_format,
15                                              tag_to_biloes="ner")
16
17      tag_type = 'ner'
18
19      tag_dictionary = corpus.make_tag_dictionary(tag_type=tag_type)
20
```

```
21      embedding_types: List[TokenEmbeddings] = [
22              FlairEmbeddings('ja-forward'),
23              FlairEmbeddings('ja-backward'),
24      ]
25
26      embeddings: StackedEmbeddings = StackedEmbeddings(embeddings=
            embedding_types)
27
28      tagger = SequenceTagger(hidden_size=256,
29                                          embeddings=embeddings,
30                                          tag_dictionary=tag_dictionary,
31                                          tag_type=tag_type,
32                                          use_crf=True)
33
34      trainer: ModelTrainer = ModelTrainer(tagger, corpus)
35
36      trainer.train(model_folder,
37                    max_epochs=150,
38                    monitor_test=True,
39                    train_with_dev=True)
```

ソースコード A.3: txt_to_bio.py

```
1       import json
2       import spacy
3
4       def create_tokens_and_labels(nlp, splitted):
5           tokens = [] # トークン格納用
6           labels = [] # トークンに対応するラベル格納用
7           for s in splitted:
8                   text = s['text']
9                   label = s['label']
10                  tokens_splitted = nlp(text)
11                  if label == "O":
12                      labels_splitted = ["O"] * len(tokens_splitted)
13                  else:
14                      labels_splitted = [f"B-{label}"]
15                      labels_splitted.extend([f"I-{label}"] * (len(
                            tokens_splitted) - 1))
16                  tokens.extend(tokens_splitted)
17                  labels.extend(labels_splitted)
```

```
18
19          return tokens, labels
20
21      def create_tagged_data(nlp, text, entities):
22          # 固有表現の前後でを分割し、それぞれのラベルをつけておく。text
23          entities = sorted(entities, key=lambda x: x['span'][0]) # 固有
                表現の位置の昇順でソート
24          splitted = [] # 分割後の文字列格納用
25          position = 0
26          for entity in entities:
27                  start = entity['span'][0]
28                  end = entity['span'][1]
29                  label = entity['type']
30                  # 固有表現ではないものにはのラベルを付与 O
31                  splitted.append({'text': text[position:start], 'label'
                        :"O"})
32                  # 固有表現には、固有表現のタイプに対応するをラベルとして
                        付与 ID
33                  splitted.append({'text': text[start:end], 'label':
                        label})
34                  position = end
35          splitted.append({'text': text[position:], 'label':"O"})
36          if splitted[0] == {'text': '', 'label': 'O'}:
37              splitted.pop(0)
38          tokens, labels = create_tokens_and_labels(nlp, splitted)
39          return {"tokens": tokens, "labels": labels}
40
41      def create_dataset(nlp, dataname):
42          # datalist = create_tagged_data(nlp, orig_data["text"],
                orig_data["entities"])
43          orig_data = txt_to_bio(f"text_{dataname}.txt")
44          with open(f"{dataname}.txt", "w", newline='\n') as f:
45              f.write("-DOCSTART-␣-X-␣-X-␣O\n\n")
46              for data in orig_data:
47                  tagged_data = create_tagged_data(nlp, data["text"],
                        data["entities"])
48                  tokens = tagged_data["tokens"]
49                  labels = tagged_data["labels"]
50                  for i in range(len(tokens)):
51                      f.write(f"{tokens[i]}\t{labels[i]}\n")
52                  f.write("\n")
53
```

```
54      def txt_to_bio(filename):
55          with open("ner.json", "r") as f:
56                  jsons = json.load(f)
57          with open(filename, "r") as f:
58                  lines = f.readlines()
59                  data = []
60                  for line in lines:
61                          for j in jsons:
62                                  if line.replace("\n", "") == j["text"
                                        ]:
63                                      data.append(j)
64          return data
65
66      nlp = spacy.load('ja_ginza')
67      name = ["train", "test", "dev"]
68      for i in name:
69              create_dataset(nlp, i)
```