Hugging Face で動かす自然言語処理入門
Introduction to Natural Language Processing with Hugging Face

嶋田和孝 [1]

[1] 九州工業大学 大学院情報工学研究院 知能情報工学研究系

1 はじめに

観光情報学会に所属する皆さんが，観光情報を扱い，何かを分析するとき，地図情報と行動履歴を融合して解析したり [1]，SNS などへの投稿頻度の時系列情報を利用したり [2]，その対象や方法はいろいろとあると思います．その中でもテキストデータは最も直感的な分析対象であるのではないかと思います．著者はコンピュータでテキストを処理する技術である自然言語処理という分野の研究をしており，この特集記事では，最近の自然言語処理で用いられる手法と Python・既存のライブラリを使った簡単な自然言語処理の実行方法について説明したいと思います．この記事の目標は，近年主流の大規模事前学習言語モデルの雰囲気を理解することと，皆さんご自身の手元にあるデータを解析する最初の一歩となることです．後半部分では，サンプルプログラムを示しながら説明しますので，皆さんが実際に現在主流のモデルを動かすことも可能です（Google Colab という環境を前提にします）．

この記事を書いているのは 2023 年の 3 月ですが，現在，ChatGPT と呼ばれる言語モデルが世界を席巻しています．その拡張版である GPT-4 もリリースされています．人間の質問に，それっぽい答えをかなり柔軟に出力してくれるシステムになっています．この ChatGPT を代表とする GPT という名前のモデルは，後述する Transformer という学習機構のデコーダ部分を使ったものです．一方で（ChatGPT の使い方を期待されている方には申し訳ありませんが），この記事では，Transformer のエンコーダ部分を使った代表格である BERT と呼ばれる大規模事前学習言語モデルを主に説明の対象にします．

ここで，重要なのは，先ほどから何度か登場している「言語モデル」という言葉です．言語モデルとは，一言で要約すると「次にどんな単語が来るか」を予測するモデルです．たとえば，「今日の天気は」の次に来る単語を想像してみてください．「晴れだ」とか「曇りだ」とか「雨だ」とかを想像すると思います．一方で「観光だ」とは思わないはずです[1]．また「1 月 1 日の今日の天気は」となれば「雪だ」の可能性も出てきますし，逆に「8 月 31 日の今日の天気は」という文脈であれば，少なくとも日本国内での話であれば「雪だ」の可能性は極めて低くなります．つまり，直前までの文脈に基づいて，次の単語が予測されるわけです[2]．これが言語モデルの基本的なイメージです．自然言語処理における言語モデルは，大量のテキストデータから，この次に出てくる単語の確率を学習しています．人間は厳密な確率で考えているわけではないですが，同じようなことを脳内でしているはずです．以降で説明される言語モデルとはこのようなことを内在している機械だと思ってください．

現在の自然言語処理は，少なくとも研究レベルでは殆どが深層学習のモデルに基づいています．GPT や BERT も深層学習のモデルです[3]．この記事では，

[1]「観光日和だ」なら大丈夫そうですけど．

[2]実際の「今日の天気」の予測しているわけではなく，単語として，一般的に何が出やすいかという予測であることに注意してください．

[3]ただし，すべての言語モデルが深層学習に基づいているわ

まず，深層学習前の自然言語処理と深層学習後の自然言語処理について簡単に説明します．その次に，前述した BERT の基本的な考え方を説明し，それ以外の大規模事前学習モデルについても軽く触れます．そのあとに，サンプルプログラムを使って，評判分析（Sentiment Analysis）を実際に動かす流れや，それ以外の事例についても説明していきたいと思います．

2　自然言語処理と深層学習

深層学習以前の自然言語処理といえば，たとえば，文書分類などでは，多くの場合，形態素解析をし，Bag-of-Words（BoW）というベクトル表現に変換したのち，Naive Bayes や Support Vector Machine などの統計的な機械学習技術を使ってモデルを作ることが多かったです．BoW に変換するときに，いかに良い特徴を捉えられるかが工夫のしどころでした．

形態素解析のあとの処理については，構文解析をして，意味解析をして... のような段階的に処理をするのが一般的でした．これは，我々が中学校や高校の英語の授業で，英日翻訳とかをするときと似ているかもしれません．まず，単語の品詞などを推定し（広義の形態素解析），主語や動詞，目的語を見つけ出し（構文解析），それぞれの単語の意味・訳語を辞書で調べ（意味解析），今度は今の流れとは逆に日本語の構文に併せて語順を変更し，活用などを考えて表層形に変換していくような作業です．

深層学習の時代になり，単語の表現は BoW ではなく，Word2Vec に代表されるような分散表現が用いられることが多くなりました．文書分類のようなタスクも深層学習によって精度が高くなることが多くありますが，深層学習によって大きく変わったのは，それまでの言語処理では苦手だった生成系のタスクで，深層学習（というかニューラルネットワーク）が機能する，という点だと思います．その代表例が機械翻訳です．前述の段階的なモデルではなく，単語を分散表現（ベクトル）に変え，入力から出力

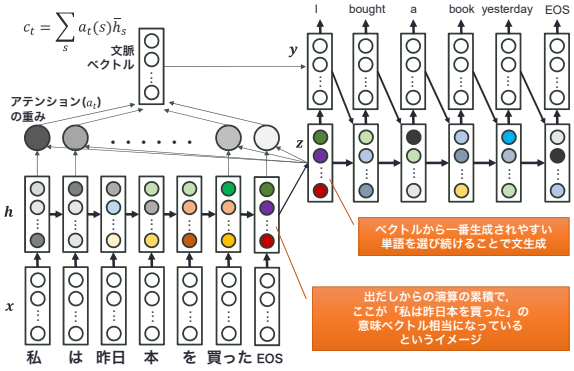


図 1: アテンション付きエンコーダ・デコーダモデル．

への射影の問題に置き換えることで，我々が中学校や高校の英文和訳で書いていた「実際には書かないであろう不自然な日本語訳」のようなものではなく，とても流ちょうな出力が可能になりました（構文的な規則を見ているわけではなく，自然な流れになる言語モデルを入出力間で直に学習しているため流ちょうになります）．このような入力から直接出力を推定するモデルのことを sequence2sequence とか，end2end，エンコーダ・デコーダモデルなどと呼びます．図 1 が機械翻訳でのイメージです．入力言語をエンコード（符号化）し，何らかの計算をしたしたあと，デコード（復号化）して言語に変換します．辞書的な意味などではなく，内部はベクトルの演算です．入力（図 1 の場合，日本語文）を読み終えたところでできているベクトルがその文の意味ベクトルのような意味になり，翻訳はそのベクトルから生成されやすい出力（この場合は英単語）を予測しながら出力していくわけです．さらに，この翻訳の場合,「I」という単語を出すときは,（人間も）「私」という単語が重要だと考えます．今，どこに着目するべきかというものを重みとしてモデルの中に組み入れたのがアテンション（注意機構）という概念です．

けではありません．単純な確率モデルもあります．

3　大規模事前学習言語モデル

2 節でのモデルは（深層学習も含む），当然ながらいくつか問題点があります．その一つは特に深層学習ベースのモデルは大量の学習データが必要，ということです．ある単語の次にどんな語が来るかを正確に予測するにはたくさんのデータが必要そうだというのは直感的に理解できるのではないかと思います．たとえば，機械翻訳をエンコーダ・デコーダモデルで実現しようとすると，数十万・数百万の翻訳ペアの事例が必要になります．日本語と英語の対になった文を 10 万文集めなさい，といわれただけで，それが非常に大変そうなのは容易に想像できると思います．その問題に対処したのがこの節で説明する大規模事前学習モデルです．

3.1　Transformer

まず，具体的なモデルの説明に行く前に，根幹の部分を簡単に説明します．真面目に説明するととても長くなりますし，数式もたくさん出てくるので，あくまでイメージとして．

以降のモデルは Transformer と呼ばれる学習機構をベースに作られています．これは，図 1 でも出てきたアテンションという機構を利用しています．Transformer は，自分自身や文中の単語間で張り巡らせた Self-Attention と Multi-Head Attention という仕組みを組み込んだエンコーダ・デコーダモデルです．なお，かなり大雑把ですが，エンコーダは入力をベクトルに変えるモデルだと考えてください．これが次節で説明する BERT です．これに対して，デコーダは文を生成するモデルだと考えると良いと思います．これが最初に出てきた ChatGPT の母体になっている GPT というモデルです．

3.2　BERT

事前学習言語モデルはたくさんありますが，以降のサンプルプログラムでは Devlin ら [3] によって提案された BERT (Bidirectional Encoder Rrepresentation from Transformers) を用います．ここでは，簡単に BERT について説明します．BERT は

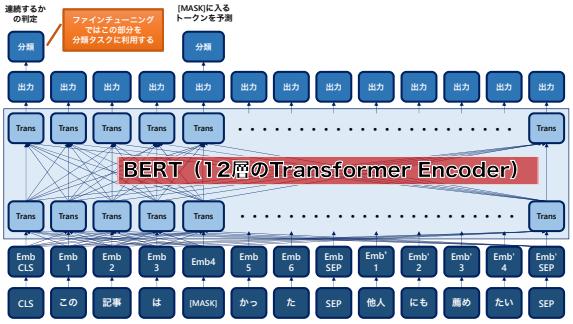


図 2: BERT のイメージ図.

大規模な教師なしテキストから汎用的な言語モデルを獲得する技術です．一般的に利用される BERT モデルは 12 層の Transformer から構成され，Masked Language Model (MLM) と Next Sentence Prediction (NSP) と呼ばれる方法によって言語モデルを学習します．

図 2 にそのイメージを示します．MLM は入力の隠された部分（MASK）にどんな単語が入るかを予測するモデルです．たとえばこの例では，最上部の MASK 部分に対応する出力として「面白」や「楽し」みたいな語が出力されることが期待されます．この文（この記事は [MASK] かった）だけを考えれば，もちろん「難し」も入りそうです．より長い文脈も学習させたいというモチベーションから，NSP が導入されています．NSP は「SEP」という特殊な記号で区切られた 2 つの文が文脈的に適切な流れになっているかを予測するモデルです．この例では，SEP の前後の文の流れは自然であるため，一番左にある CLS という入力に対応する出力に「連続する」という意味を表す「1」を出力することが期待されます（不自然な文の場合は 0 を出力します）．このように，BERT は文内での単語の繋がり（MLM）や，文をまたいだ関係性（NSP）を保持した言語モデルとなっています[4]．

大量のデータから MASK に何が入るかを学習したり，NSP の値を推定したりすることで得られる言語モデルは，特定の目的やタスクに依存しない汎

[4] その結果，この文脈では，前述の例の「難し」の確率は低くなることが期待されます．

用的なモデルになっていることが期待されます．一方で，何か特定のタスクを解きたいとき，そのタスクに特化したモデルのほうが性能が良いはずです．BERT では，この汎用的な言語モデルに対して，対象となる学習データを適用し，そのタスク・データに特化した分類モデルを容易に作ることができます．これをファインチューニングと呼びます．たとえば，次節（4 節）では，文が肯定的な意見か否定的な意見かを分類する評判分類タスクを取り上げますが，この場合，事前に肯定的な文なら 1，否定的な文なら 0 のラベルがついたデータなどを用意して，素の BERT を評判分類タスク用にファインチューニングしたモデルを利用しています．

旧来の機械学習モデルでは，学習データが大量に必要でした．これは，その解きたいタスクに対する知識だけではなく，データ中の言語に関する情報も同時に学習する必要があるからです．一方で，BERT のような事前学習モデルは，その言語に関する基本的な知識を大量のデータからすでに言語モデルとして学習しているため，解きたいタスクのデータは少なくても上手くいくことが多いというメリットがあります（当然，学習データが多い方が精度が上がる傾向はあります）．誤解を恐れず言うなれば，我々がたとえば何かの試験を受けるときに，日本語を一から学びつつ，その模擬問題集で学習しているのではなく，すでに身につけている日本語の知識を使って，模擬問題集で試験用に脳みそをチューニングするのと同じような感覚です．さらに，BERT 自体の学習は既存のテキストデータを勝手に所々[MASK] に置き換えて予測する問題を解けば良いので，とりあえず，どこかからテキストを集めてくれば良いわけです．本節の冒頭で述べたように，数十万・数百万の翻訳ペアを苦労して作る必要はありません（先ほど説明したように，ファインチューニングするときは当然データが多い方が良いわけですが）．これも事前学習言語モデルのメリットです．

3.3 他の言語モデル

分類問題を解く場合は，BERT を使うのが最初の一手だと思います．BERT はエンコーダであるため，基本的に生成のためのモデルではありません．生成をしたり，対話システムを作ろうとする場合はデコーダ部分も必要です．以下に有名どころを簡単に説明します．

- GPT：Transformer のデコーダ部分を使っている言語モデルです．GPT，GPT-2，GPT-3，GPT-4 と数字が上がるごとに新しく，モデルが巨大になっています．GPT-3 では，BERT のようにファインチューニングするのではなく，適切な事例（プロンプトと呼ばれます）を与えると，少量の事例（Few-Shot learning）でも上手くいくことが知られています．ChatGPT はこの GPT をベースに人間が追加で質問応答のペアデータを大量に作って強化学習をしたモデルです[5]．

- T5：Text-to-Text Transfer Transformer と呼ばれているモデルです．Transformer のエンコーダ・デコーダモデルです．すべてをテキストからテキストへの変換問題として学習します．たとえば，BERT だと，分類問題を解くときは，文を入力すると，図 2 で一番左上にある「分類」と書かれた部分（CLS トークンと呼ばれます）のベクトルを使って，既存のラベルからどれが最もらしいかを予測するのですが，T5 の場合，そういうラベルを予測するのではなく，テキストとして出すように学習されています．

- BART：これもエンコーダ・デコーダモデルです．シンプルに説明すると BERT（エンコーダ）と GPT（デコーダ）が組み合わさったようなモデルです．入力を BERT と同じように MLM で学習したり，部分的に単語を除いたり

[5]なお，事実とは異なる出力をすることがありますが（Hallucination と呼ばれる問題です），これは言語モデルであるという特性上当然の結果です．たとえば，次の電車の時間を質問しても，そのタイミングで検索しているわけでなく，学習済みの言語モデルから次に出やすい単語を予測しているだけなので．

して学習します．その変換された文を元の文に戻すようにデコーダが学習をするという枠組みです．BERT のようなエンコードができるため分類問題も対応できますが，GPT のような生成もできるため，要約や翻訳などの生成タスクでよく用いられているように思います．

4 Hugging Face で動かしてみる

この記事では，環境として Google Colab[6] を利用します．Google Colab は Google のアカウントがあれば使えるはずです．実装には Hugging Face[7] と呼ばれる機械学習用のプラットフォームを用います．Hugging Face には 3 節で説明した Transformer をベースとした深層学習モデルが多く実装されており，それを呼び出すだけで使うことができます．今回はこの中にある pipeline という機能を使って，評判分析などを題材に，実際にプログラムを動かしてみましょう．

サンプルプログラムは以下の URL にあります．

`http://www.pluto.ai.kyutech.ac.jp/~shimada/STI/sti_code.zip`

この中には 2 つのファイルがあります．内容は同じです．

- `STI_code.ipynb`：Google Colab にアップロードするだけで実行できます．
- `STI_code.py`：ご自身の PC などで Python のコードを実行する場合に使ってください．本記事の最後の付録としてついているのがこちらです．

Google Colab を起動すると図 3 のような画面になるはずです．「ノートブックを新規作成」を選んで `STI_code.py` からコピペしながら実行しても良いですし，「アップロード」を選んで `STI_code.ipynb` をアップロードしても良いです．Google Colab では，各コードの左側にある再生マークを押すと実行されます（図 4 を参照）．

[6] `https://colab.research.google.com/`
[7] `https://huggingface.co/`

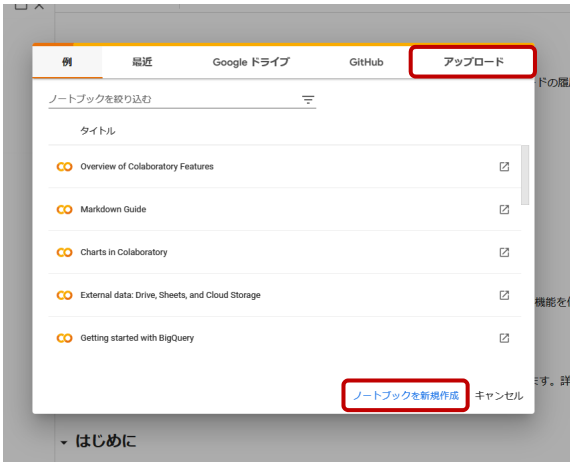


図 3: Google Colab の起動．

図 4: Google Colab でのプログラムの実行．

上から順番に実行していきましょう．最初の[8]

`!pip install ...` (2)

は実行に必要な要素（特に日本語関係のもの）を一括で読み込むための命令です[9]．次の

`from transformers import ....` (5)

で前述した pipeline という機能や BERT を実行するためのライブラリを読み込んでいます．次にトークナイザとモデルを設定します．

`tokenizer = BertJapaneseTokenizer....` (8)

`model1 = AutoModelForSequenceClassification....`

トークナイザとは文を BERT が扱えるように単

[8] なお，以降プログラムの行末の数字は，付録のプログラムでの行番号を意味しています．つまり，以下の (2) は 2 行目という意味です．
[9] 実行が終わるまで 1〜2 分かかることもあります．気長に待ってください．

語のような単位[10]に区切るための道具です．モデルで利用しているものと同じトークナイザを指定します．ここでは東北大学の公開しているものを利用しています[11]．モデルについては，Hugging Face で公開されているものを探してきて，適用します．あとで別のモデルも試すため，ここではこのモデルを model1 と名付けています．このモデルは，肯定（Positive）・否定（Negative）の 2 つだけではなく，中立（Neutral）の 3 つの分類が可能なようです[12]．このトークナイザとモデルを利用して処理をします．今回は評判分析（Sentiment Analysis）をしますので，pipeline で sentiment-analysis を指定し，先ほどのトークナイザとモデルを引数として渡します．

```
classifier1 = pipeline("sentiment-analysis",
 model=model1,tokenizer=tokenizer)
```
(12)

これで準備は完了です．次の

```
result = classifier1("...")
```
(16)〜

```
print(result)
```

のところで，判別したい文を classifier1 に渡して，肯定・否定・中立の判断をして，その結果を出力しています．図 5 のような出力になるはずです．列挙すると以下の通りです．

- UI が素晴らしいと思う。直感的に操作できるので操作方法を調べなくても利用できる
  判定：POSITIVE　スコア：0.97608...
- 価格は普通だと感じた。
  判定：NEUTRAL　スコア：0.83697...
- 機能に対して利用料金が高すぎると思う。また、クラウドに文章データを送るのでセキュリティも不安に思っている
  判定：NEGATIVE　スコア：0.98796...

これは 1 つめの文章が POSITIVE と判断され，その度合いが 0.976... であることを意味しています．

[10]正確には単語ではなく，サブワードと呼ばれる文字と単語の中間のような単位に分割されます．
[11]`cl-tohoku/bert-base-japanese-whole-word-masking`
[12]`koheiduck/bert-japanese-finetuned-sentiment`

```
[ ]  # 設定した classifier1 を使って引数に渡した文の判定
     # このモデルは肯定・否定・中立の判断をする
     result = classifier1("UIが素晴らしいと思う。直感的に操作できるので操作方法を
     print(result)
     result = classifier1("価格は普通だと感じた。")
     print(result)
     result = classifier1("機能に対して利用料金が高すぎると思う。また、クラウドに
     print(result)

     [{'label': 'POSITIVE', 'score': 0.9760897159576416}]
     [{'label': 'NEUTRAL', 'score': 0.8369797468185425}]
     [{'label': 'NEGATIVE', 'score': 0.9879699945449829}]
```

図 5: 各文章の肯定・否定・中立度合いの出力．

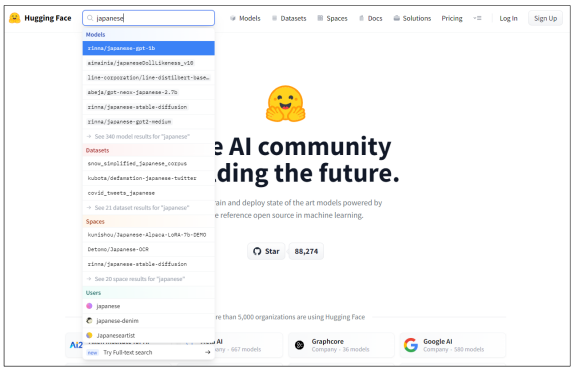


図 6: モデルを検索．

残り 2 つの結果もなんとなく妥当な結果が出ている気がします．

`AutoModelForSequenceClassification` で別のモデルを設定すれば，当然，結果は変わってきます．精度や分類項目はモデルを公開した人がどんなデータでファインチューニングしたかで異なるからです．サンプルプログラムの model2[13]では肯定と否定の二値分類になっています．

どんなモデルがあるかは Hugging Face の Web ページで検索できます．図 6 では，検索窓に「japanese」と入力した例です．キーワードを組み合わせて，ご自身の問題設定に沿ったモデルを探してみると良いでしょう．なお，Hugging Face に公開されているモデルが作者などによって削除されると使うことができなくなる点に注意してください．

他のモデルや pipeline でタスク指定をすることで，評判分析以外も実行可能です．

[13]`abhishek/autonlp-japanese-sentiment-59363`

```
seq2seq = pipeline("summarization", ...
```
(37)

で始まる部分は T5 をベースに要約用にファインチューニングされたモデルを利用し，pipeline で summarization を指定した例です．元の文が短くなって出力されていることが分かります．

ここまでは分析したい文を直接プログラムに書いていましたが，実際には分析したいデータファイルがあり，それを読み込んで処理したい場面も多いと思います．Google Drive にアップロードされたファイルを読み込んで前述の classifier1 で処理するプログラムを見てみましょう．Google Colab でご自身の Google Drive のファイルを参照するには，Colab 側でマウントする必要があります．手順を図 7 に示します．Google Colab のページで左側にファイルに関する情報を表示します．図 7(a) に示すようなファイルに関する情報が出ていなければ，一番左にあるいくつかのアイコンのうちフォルダのようなかたちをしたアイコンをクリックしてください．その後，図 7(a) で示すように Google Drive にマウントするためのアイコンをクリックします．そうすると，図 7(b) のような画面になり，アクセス許可のためのポップアップが開きます．アクセスを許可し，ご自身の Google アカウントと紐付けてください．マウントが終わると図 7(c) の左側のように [drive] → [MyDrive] というフォルダができ，ご自身の Google Drive のホームディレクトリが見えるようになるはずです．この状態になれば，Python で普通のファイルを開くような感覚でファイルをオープンすることができます[14]．

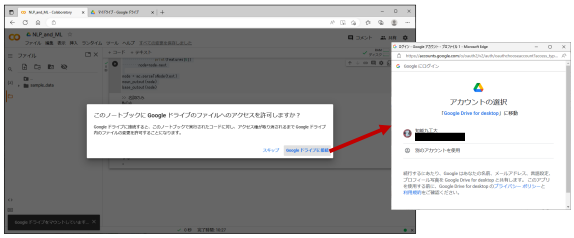
```
with open("/content/drive/...
```
(50)

から始まるプログラムは，Google Drive のホームディレクトリにある `sentences.txt` というファイルを読み込み，1 行単位で評判分析するプログラムです．

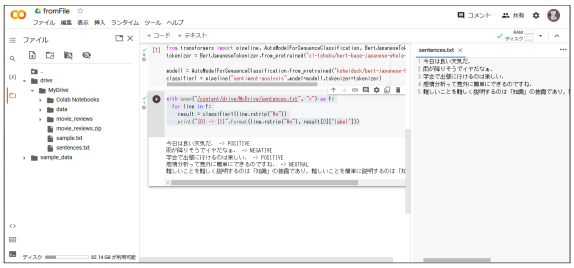
pipeline でどのようなことができるのかをすべて説明する余裕はないので，皆さんそれぞれで調べてください．観光情報の分析でいうと，評判分析以外にも固有表現（地名やモノの名前など）を自動抽出

[14] パスは `/content/drive/MyDrive/`

(a)



(b)



(c)

図 7: Google Drive を Google Colab にマウント．

したい場合も多いかもしれません．

```
model_name = "tsmatz/xlm-roberta....
```
(56)

からはじまるプログラムは，固有表現抽出の pipeline の実装例です．これは BERT の亜種である RoBERTa というモデルをベースに固有表現抽出タスクにファインチューニングされたものです．このプログラムを実行すると

- 先日，北海道で開催された観光情報学会に参加して，「白い恋人」と「じゃがポックル」を買って帰った

という入力に対して図 8 のような結果が返ってきま

```
[{'entity': 'LOC', 'score': 0.99747026, 'index': 5, 'word': '北海道', 'start': 3, 'end': 6},
 {'entity': 'EVT', 'score': 0.9940996, 'index': 9, 'word': '観光', 'start': 12, 'end': 14},
 {'entity': 'EVT', 'score': 0.99292654, 'index': 10, 'word': '情報', 'start': 14, 'end': 16},
 {'entity': 'EVT', 'score': 0.99328846, 'index': 11, 'word': '学会', 'start': 16, 'end': 18},
 {'entity': 'PRD', 'score': 0.99945503, 'index': 16, 'word': '白', 'start': 25, 'end': 26},
 {'entity': 'PRD', 'score': 0.99947244, 'index': 17, 'word': 'い', 'start': 26, 'end': 27},
 {'entity': 'PRD', 'score': 0.9995074, 'index': 18, 'word': '恋人', 'start': 27, 'end': 29},
 {'entity': 'PRD', 'score': 0.99956614, 'index': 22, 'word': 'じゃ', 'start': 32, 'end': 34},
 {'entity': 'PRD', 'score': 0.99953246, 'index': 23, 'word': 'が', 'start': 34, 'end': 35},
 {'entity': 'PRD', 'score': 0.99958676, 'index': 24, 'word': 'ポ', 'start': 35, 'end': 36},
 {'entity': 'PRD', 'score': 0.9995521, 'index': 25, 'word': 'ック', 'start': 36, 'end': 38},
 {'entity': 'PRD', 'score': 0.99947625, 'index': 26, 'word': 'ル', 'start': 38, 'end': 39}]
```

図 8: 固有表現抽出の結果.

す．たとえば,「北海道」は LOC（場所）でその確信度は 0.9974 であったり,「観光」と「情報」と「学会」は EVT（イベント）でそれぞれ文中の 12 文字目，14 文字目，16 文字目から始まる（end は実際の位置 −1 の値）などを意味しているのですが，このままでは少し人間にとっては読みづらいです．このような出力を整形し，
['LOC:北海道', 'EVT:観光情報学会', 'PRD:白い恋人', 'PRD:じゃがポックル']
のようなかたちで出力するプログラム（getNEs()）も付けていますので，参考にしてください．

5 おわりに

この記事では，近年主流の大規模事前学習モデルに基づく自然言語処理入門（今に至るまでの経緯も含め）という立ち位置で説明してきました．Hugging Face を利用することで，比較的簡単に実装ができる便利な世の中になりました．2 節や 3 節で説明した中身の話と 4 節で示したサンプルコードとその説明が，皆さんの一助にでもなればと思います．

当初は手持ちのデータでファインチューニングするところまで含めようかと考えていましたが，さすがにそれは分量が多くて断念しました．より詳しい話やより正確な説明は書籍や Web の記事などが数多くありますので，そちらに譲ります．BERT や自然言語処理に関する実装や説明については [4] や [5] などが分かりやすいように思います．より深く自然言語処理や Transformer などの理論的な背景を学びたい方は [6] がもっとも充実している書籍（教科書）だと思います．

一見，深層学習や最新の機械学習技術を用いた分析は華やかで，何でもできそうな気になってきますが，実際には地道な分析が欠かせないと思います．今回の特集記事では．辞書の作り方やその応用例などを示した徳久による記事があります [7]．深層学習や機械学習は多くの場合，分析のための道具（手段）であり，目的ではありません．実際にはデータを見て，データを理解し，道具としての自然言語処理や機械学習を有効利用することが重要です．共同で書いた書籍[15]ですが，[8] のようなものも皆さんが何かを分析する際の参考になるかもしれません．

残念ながら，現状では，実行ボタンを押したら，ポンと期待した結果が出てくる夢の技術はまだありません．それでも，既存の自然言語処理の技術が，皆さんの目的に合った道具として有効に機能することを祈っています．

参考文献

[1] 田村和範, 笠原秀一, 柿本雅之, 美濃導彦．プローブカーデータと停車エリア間類似度を用いた旅行行動のモデル化．観光情報学会「観光と情報」, Vol. 10, No. 1, pp. 101–112, 2014.

[2] 嶋田和孝, 上原尚, 遠藤勉．集合知に基づく観光地推薦システムの構築．観光情報学会「観光と情報」, Vol. 10, No. 1, pp. 113–124, 2014.

[15] 電子情報通信学会・言語理解とコミュニケーション（NLC）研究会の幹事団で書きました．

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. 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. 4171–4186, 2019.

[4] 佐藤大輔, 和知徳磨, 湯浅晃, 片岡紘平. BERT 入門 プロ集団に学ぶ新世代の自然言語処理. リックテレコム, 2022.

[5] 新納浩幸. PyTorch 自然言語処理プログラミング. 株式会社インプレス, 2021.

[6] 岡崎直観, 荒瀬由紀, 鈴木潤, 鶴岡慶雅, 宮尾祐介. 自然言語処理の基礎. オーム社, 2022.

[7] 徳久雅人. 観光地分析のための感情辞書および観光行動辞書の作成. 観光情報学会誌 観光と情報, Vol. 19, No. 1, 2023.

[8] 榊剛史, 石野亜耶, 小早川健, 坂地泰紀, 嶋田和孝, 吉田光男. Python ではじめるテキストアナリティクス入門. 講談社サイエンティフィク, 2022.

著者情報

嶋田和孝．1997 年大分大学工学部 知能情報システム工学科卒業．1999 年同大学大学院 博士前期課程修了．2002 年同博士後期課程 単位取得満期退学．同年より九州工業大学情報工学部 知能情報工学科 助手．現在，大学院情報工学研究院 知能情報工学研究系 教授．評判分析をはじめとする Web 上のデータの分析や知識獲得の研究に従事．現在は人間同士の対話理解や議論支援の研究を行っている．観光情報学会，言語処理学会，電子情報通信学会，情報処理学会，人工知能学会各会員．

付録：プログラム

ソースコード 1: プログラム（前半）

```
# 関連するファイルを一式インストール
!pip install transformers['ja']

# 各種ライブラリを import
from transformers import pipeline, AutoModelForSequenceClassification,
    BertJapaneseTokenizer,BertTokenizer, BertForSequenceClassification

# トークナイザーとモデルを設定
tokenizer = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-japanese-whole-
    word-masking')
model1 = AutoModelForSequenceClassification.from_pretrained('koheiduck/bert-japanese-
    finetuned-sentiment')

# パイプラインで評判分析(sentiment analysis) を設定
classifier1 = pipeline("sentiment-analysis",model=model1,tokenizer=tokenizer)

# 設定した classifier1 を使って引数に渡した文の判定
# このモデルは肯定･否定･中立の判断をする
result = classifier1("
    UI が素晴らしいと思う。直感的に操作できるので操作方法を調べなくても利用できる")
print(result)
result = classifier1("価格は普通だと感じた。")
print(result)
result = classifier1("機能に対して利用料金が高すぎると思う。また、クラウドに文章データを
    送るのでセキュリティも不安に思っている")
print(result)

# 別のモデルを読み込み,同じく sentiment analysis を pipeline で設定し,
# classifier2 という名前を付ける
model2 = AutoModelForSequenceClassification.from_pretrained('abhishek/autonlp-japanese-
    sentiment-59363')
classifier2 = pipeline("sentiment-analysis",model=model2,tokenizer=tokenizer)

# 設定した classifier2 を使って同様に判定
# このモデルは肯定･否定判定のみ
result = classifier2("
    UI が素晴らしいと思う。直感的に操作できるので操作方法を調べなくても利用できる")
print(result)
result = classifier2("価格は普通だと感じた。")
print(result)
result = classifier2("機能に対して利用料金が高すぎると思う。また、クラウドに文章データを
    送るのでセキュリティも不安に思っている")
print(result)
```

ソースコード 2: プログラム（後半）

```
# 要約用にファインチューニングされたT5 のモデルを読み込み，要約処理
seq2seq = pipeline("summarization", model="tsmatz/mt5_summarize_japanese")
sample_text = "サッカーのワールドカップカタール大会、世界ランキング 24位で
グループE に属する日本は、23 日の 1 次リーグ初戦において、世界 11 位で過去 4 回の
優勝を誇るドイツと対戦しました。試合は前半、ドイツの一方的なペースで
はじまりましたが、後半、日本の森保監督は攻撃的な選手を積極的に動員して
流れを変えました。結局、日本は前半に 1点を奪われましたが、途中出場の
堂安律選手と浅野拓磨選手が後半にゴールを決め、2対 1で逆転勝ちしました。
ゲームの流れをつかんだ森保采配が功を奏しました。"
result = seq2seq(sample_text)
print(result)

# google drive をマウントし，ファイルを読み込んで処理する
# 使うモデルは classifier1
with open("/content/drive/MyDrive/sentences.txt", "r") as f:
  for line in f:
    result = classifier1(line.rstrip("\n"))
    print("{0} -> {1}".format(line.rstrip("\n"), result[0]['label']))

# 固有表現抽出タスクを実行
model_name = "tsmatz/xlm-roberta-ner-japanese"
classifier = pipeline("token-classification", model=model_name)
result = classifier("先日,北海道で開催された観光情報学会に参加して,「白い恋人」と「じゃがポックル」を買って帰った")
print(result)

# 成形して,固有表現単位で出力
def getNEs(result):
    word = ""
    prevTag = ""
    prevEnd = -1
    data = []
    for item in result:
        if (item['word'] == '?'):
            continue
        if (item['entity'] == prevTag and item['start'] == prevEnd):
            word += item['word']
        if (item['start'] != prevEnd):
            if (word != ""):
                word = prevTag + ":" + word
                data.append(word)
            word = item['word']
        prevTag = item['entity']
        prevEnd = item['end']
    word = prevTag + ":" + word
    data.append(word)
    return(data)

print(getNEs(result))
```