令和 2 年度茨城大学大学院理工学研究科情報工学専攻　修士

学位論文

ラベル付き文集合を用いた事前学習済み BERT モデルの評価

所属　　情報工学専攻

著者　　芝山直希 (19NM714T)

指導教員　　新納浩幸教授

令和 3 年 2 月 5 日 (金)

令和 2 年度茨城大学大学院理工学研究科情報工学専攻　修士学位論文

ラベル付き文集合を用いた事前学習済み BERT モデルの評価

著者

芝山直希 (19NM714T)

指導教員

新納浩幸教授

論文要旨

本論文では事前学習済み BERT モデルに対し、ラベルが付与されている文で構成されるデータセットを入力し、その出力が内包する各文の CLS トークンの埋め込み表現を利用した評価手法を提案する。また、 提案手法による評価、空所推定タスクを用いた評価の 2 つのアプローチを用いて BERT モデルを比較し、比較結果について考察する。

2018 年に事前学習型の機械学習モデル Bidirectional Encoder Representations from Transformers (BERT) が提案された。Transformer を Encoder として活用し文全体から文脈を学習できるこのモデルは、自然言語処理の様々なタスクの精度向上に貢献している。BERT のような事前学習モデルを評価・比較する場合、一般的にはタスクベースのアプローチが取られる。しかし、このアプローチを用いた評価方法はファインチューニングを必要とする場合が多く、評価値にファインチューニングのメタパラメータが影響する可能性がある。また、公開されている BERT モデルには日本語をはじめとする「標準的な評価用データセットが存在しない言語」向けに事前学習されたものも存在する。これらのモデルに対し適切に評価し優劣を判断するためにはどのようなアプローチをとれば良いのだろうか。

本論文では現在公開されている 7 つの日本語向けの事前学習済み BERT モデルを評価対象とする。評価対象である各モデルに対し、Livedoor ニュースコーパスに対する CLS トークンの埋め込み表現群から計算した評価値、及び日本語で記述された Amazon レビュー文書から作成した空所推定データセットに対する精度の 2 つの指標で評価・比較した。提案手法を利用した比較結果は空所推定タスクの結果とは異なる傾向を有することが判明した。

Master's Thesis in Scholastic 2021, Major in Computer and Information Sciences,
Granduate School of Science and Engineering, Ibaraki University

An Evaluation Method for Pretrained BERT Model with Labeled Sentences

Auther

Naoki Shibayama (19NM714T)

Adviser

Prof. Hiroyuki Shinnou

Abstract

In this thesis, I propose the evaluation method for pre-trained BERT models with labeled sentences: input sentences to model and use [CLS] tokens from outputs. And I compare BERT models with 2 approach: evaluation with my method and evaluation with fill mask task, and discuss about the result of comparison.

Pre-training type machine learning model which name was Bidirectional Encoder Representations from Transformers (BERT) was proposed in 2018. It uses Transformer as Encoder, so BERT can learn context from whole sentence and help in the inprovement of the performance of natural language processing tasks. For evaluating or comparing pre-training models like BERT, task-based approaches were generally adopted. However, most of evaluation methods with this approach are required fine-tuning, so there is a possibility that meta parameters for fine-tuning influence results of the evaluations. And pre-trained models for languages that has no standard evaluation dataset like Japanese. Which approach is the best way for evaluating and comparing those models right?

In this thesis, evaluation targets are 7 released pre-trained Japanese BERT models. I evaluated and compared models with 2 indexes: score that calculated from CLS embeddings for Livedoor news corpus and accuracy of fill mask task dataset that was made from Amazon review documents written in Japanese. I found that the result of evaluation with my method had different tendency from the result of evaluation with fill mask task.

目次

第 1 章

序論

2018 年、事前学習型の自然言語処理モデルである BERT (Bidirectional Encoder Representations from Transformers) [1] が提案された。初版発表時でも多くの自然言語処理タスクで SoTA を達成したこのモデルは、BERT を元にした後継のモデルとともに自然言語処理の様々なタスクの精度向上に貢献している。さて、BERT をはじめとする事前学習型の自然言語処理モデルを評価・比較する場合、一般的にはタスクベースのアプローチが取られる。このアプローチを用いた事前学習済みモデルの評価工程にファインチューニングが含まれることが多く、ファインチューニング時のメタパラメータが評価値に影響する可能性がある。また、事前学習時に使用されたデータセットの言語によっては評価用データセットの問題が発生する。英語向けの場合ならば、標準ベンチマークである GLUE (General Language Understanding Evaluation) [2] や質問応答の評価用データセット SQuAD (Standard Question Answering Dataset) [3] を使えば良い。しかし、他の標準的な評価用データセットがない言語向けのモデルを評価・比較対象とする場合どのようなデータセット・アプローチを選択するのが適切だろうか。

本論文では京都大学の黒橋・村脇研究室が公開している BASE 通常版・LARGE WWM 版モデル[*1]、森長氏が公開しているモデル[*2]、Yohei Kikuta 氏が公開しているモデル[*3]、東北大学の乾・鈴木研究室が公開しているモデル[*4]、情報通信研究機構 (NICT) が公開

[*1] http://nlp.ist.i.kyoto-u.ac.jp/index.php?BERT 日本語 Pretrained モデル
[*2] https://qiita.com/mkt3/items/3c1278339ff1bcc0187f
[*3] https://github.com/yoheikikuta/bert-japanese
[*4] https://github.com/cl-tohoku/bert-japanese

しているモデル[*5]、株式会社 Laboro.AI が公開しているモデル[*6]の 7 つを評価・比較の対象とする。各モデルに対し、Livedoor ニュースコーパスの記事タイトルから生成した CLS トークンの埋め込み表現及びカテゴリラベルを利用して算出した評価値 [4]、及び Webis-CLS-10 の日本語ドメインのデータ内の文書から作成したデータセットを用いた空所推定タスク [4] の 2 つのアプローチでの評価・比較を行う。

[*5] https://alaginrc.nict.go.jp/nict-bert/index.html

[*6] https://laboro.ai/column/laboro-bert/

第 2 章

関連研究

本章では、本論文を理解する上での前提となるいくつかの研究について述べる。

2.1 BERT

正式名称を Bidirectional Encoder Representations from Transformers というこのモデルは、事前学習型の自然言語処理モデルである。後述する Transformer [5] を Encoder に採用したことで、双方向での学習を実現した。大規模データセットを用いて事前学習が行われたモデルに対し、実際に運用するタスクに対するファインチューニングを行うのが基本的な使用方法である。事前学習時に用いられるタスクは複数の空欄 ([MASK] トークン) が含まれる文に対し空所推定を行う Masked Language Model 、入力された 2 文が連続しているかどうか判定する Next Sentence Prediction の 2 つである。BERT の事前学習の概要を図 2.1 に示す。

Masked Language Model では入力文の一部を [MASK] トークンや別の単語のトークンに置換し、置換前の入力文のトークンをラベルとして使用する。確率で [MASK] トークンへの置換を行わない、性能向上のために別の単語のトークンに置換する等の処理が置換時に行われる。また、置換の単位をトークンから単語へと変更し、対象となった単語のサブワード[*1]を含めた「単語全体」を [MASK] トークンに置換する Whole Word Masking という手法が採用される場合がある。このタスクを用いて単語トークンが入力された際の出力を最適化する。Next Sentence Prediction では [SEP] トークンで区切られた 2 文を入力し、それらが連続した文であるかどうかを識別する。事前学習時におい

[*1] ##から始まるトークンを指す。

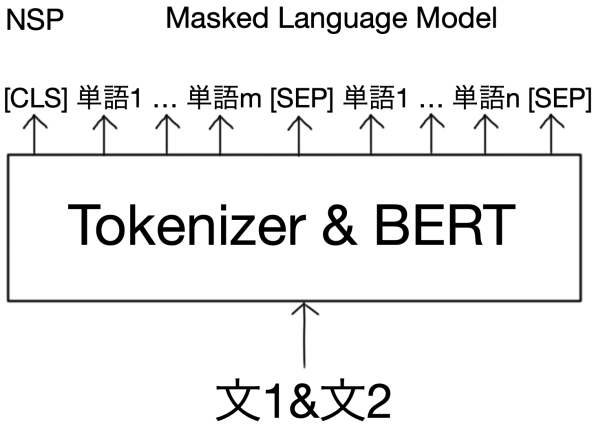

図 2.1: BERT の事前学習 (概要)

て、[CLS] トークンはこのタスクに最適化される。各タスクの概要を図 2.2 及び図 2.3 にそれぞれ示す。

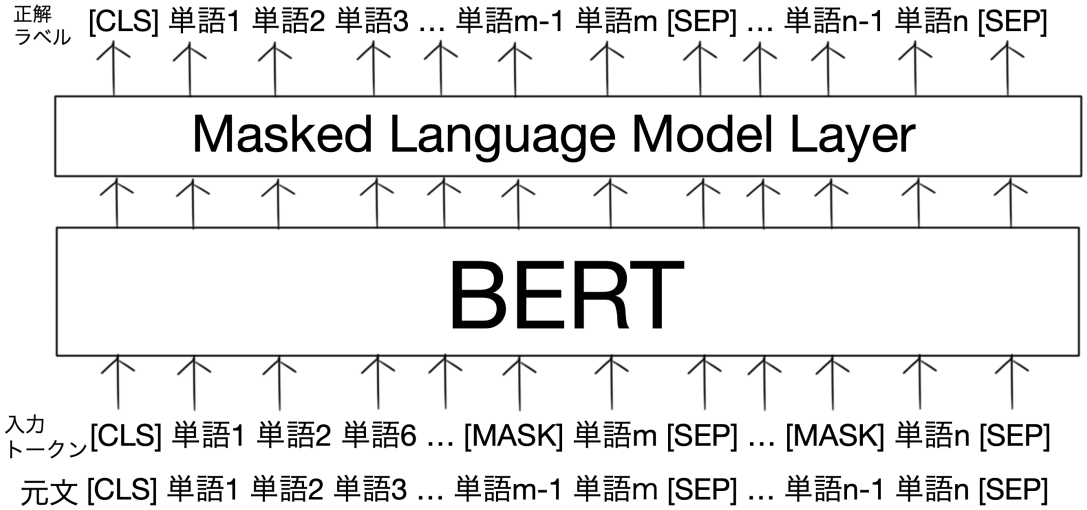

図 2.2: Masked Language Model タスク (概要)

本論文では日本語のデータを用いて事前学習が行われた BERT モデルを評価対象と

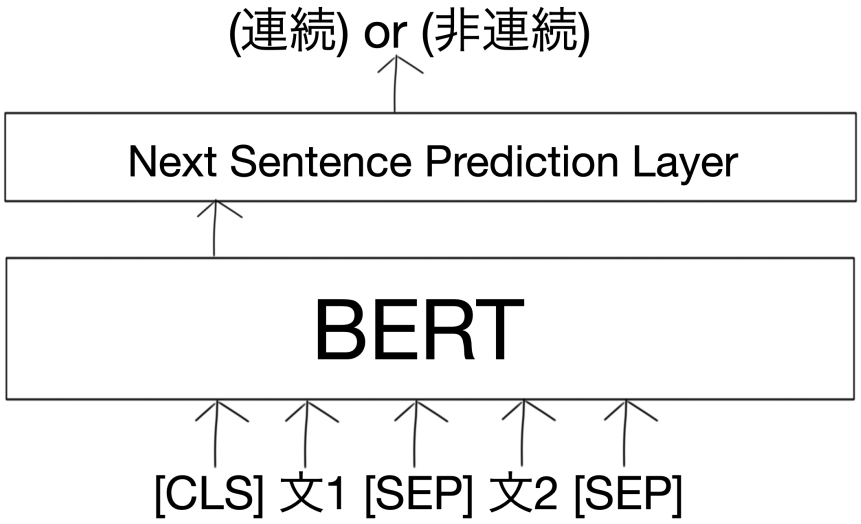

図 2.3: Next Sentence Prediction タスク (概要)

した。

2.2　Transformer

Transformer は Attention ベースの Encoder-Decoder 型モデルである。RNN 、CNN を一切使用せず、 Attention のみで構成されているモデルであり、特徴的な構造として挙げられるのが、小規模の Attention を一定数作成・計算しそれらを結合することで Attention を計算する Multi-Head Attention である。また、並列化しやすく、 RNN や CNN を採用しているモデルに比べ訓練時間が短いという特徴がある。

本論文で評価対象とした BERT をはじめとする多くの強力な自然言語処理モデルは、この Transformer をベースとしている。

2.3　埋め込み表現の評価

埋め込み表現の評価は事前学習済みモデルの場合と同様にタスクベースでの評価を行うしかない。一方、評価したい埋め込み表現に対し、人間が手作業で付与した類似度と埋め込み表現から導出した単語間との類似度との相関を測り評価した [6] 研究がある。こ

の研究では word2vec [7] などが出力する「単語の埋め込み表現がその単語の意味を表現している」という観点に基づいており、本論文の提案手法はこの観点を利用したものである。

第 3 章

タスクベースでの評価

タスクベースで事前学習済みモデルを評価する際の敷居は低い。ファインチューニング後テストデータに対する精度を測定し、比較するのが最も単純なタスクベースでの評価方法である。さて、評価を行うのに適しているタスクは何だろうか。

本章では、モデルの評価に使用できるタスク及び標準的な評価用データセットについて触れる。

3.1 文書分類

文書分類とは、複数の文を含む文書を分類するタスクである。訓練データから主に以下の事項を学習し、その内容はモデルによる推論に影響する。

- 何クラスに分類するか
- 各クラスのデータの傾向及び識別方法

このタスクを事前学習済み BERT モデルの評価に使用する場合、 BERT は入力が文書であることを想定していない点に考慮する必要がある。現時点においてこの問題点に対する標準的な対処方法は確認できておらず、評価に適したタスクであると言うことは出来ない。本論文の前身となった研究では、[CLS] トークンの値を用いた評価及び文書内の全単語トークンの平均値を用いた評価を行うことで、この問題点に対する対応を図り評価用タスクとして使用した [8]。

3.2　空所推定

空所推定とは、空欄 ([MASK] トークン) にあてはめられる単語 (トークン) を予測するタスクである。モデルは各空所が候補となった単語である確率を出力し、それを用いて評価する。このタスクは BERT の事前学習に用いられる Masked Language Model に類似しており、ファインチューニングを行わずとも一定以上の精度で予測することが期待できる。

本論文では文書分類用データセットから空所推定のデータセットを構築し、評価に使用した。

3.3　感情分析

感情分析とは、入力された文或いは文書が肯定的なものか否定的なものかを識別するタスクである。多くのデータセットでは二値ラベルが付与されていることが多く、モデルは入力文 (または文書) を用いた二値分類タスクを解くことになる。その際の入力が複数の文からなる文書であった場合、事前学習済み BERT モデルの評価においては文書分類タスクと同様の問題を抱えることになる。

本論文の前身となった研究で使用したデータセットはこのタスクを想定したものである。また、前節で述べた空所推定のデータセットの元になった文書分類用データセットと同一のものである。

3.4　質問応答

質問応答とは、質問文に対する回答を資料文から探索し、回答するタスクである。モデルには質問文と回答の根拠として使用する「資料文」のペアが入力される。モデルは資料文から正解単語を探索し、回答する。ただし、データセットによっては質問文に対する正解が資料文中に存在しないことがある。その場合、モデルは正解が存在しないことを出力しなければならない。

本論文では評価用のタスクとして使用していない。なお、SQuAD はこのタスクの英語用のデータセットである。また、後述する GLUE には資料文中に質問文に対応する正解があるかどうかを識別するタスクのデータセットが含まれている。

3.5　自然言語推論

自然言語推論 (Natural Language Inference, NLI) とは、ある一文を前提とした場合において仮説文が成立しているかを識別するタスクである。三値分類タスクであり、モデルは前提文と仮説文のペアから「前提から仮説が推論できる」「仮説が前提と矛盾する」「どちらでもない」のいずれかのラベルに対応する値を出力する。

本論文では評価用のタスクとして使用していない。なお、後述する GLUE ベンチマークにはこのタスクのデータセットが含まれている。

3.6　GLUE

対象となる自然言語が英語である場合、標準的なベンチマークである General Language Understanding Evaluation (GLUE) ベンチマークを利用できる。これは主に言語理解を必要とする分類・質問応答などの形式のデータセット群からなる。その一覧を表 3.1 示す。

本論文において評価対象とするモデルは日本語で事前学習が行われているため、このベンチマークを実施出来ない。

表 3.1: GLUE タスク一覧

データセット (略称)	種別	概要
CoLA	分類	文法として正しいかどうか識別する。
SST-2	分類	感情分析タスクのデータセット。
MRPC	分類	二文が同じ意味であるかどうか識別する。
STS-B	回帰	二文の類似度を判断する。
QQP	分類	二つの質問文が同じ意味であるか識別する。
MNLI	分類	二文の含意関係を推定する。Matched (MNLI-m) と Mismatched (MNLI-mm) の二種がある。
QNLI	分類	資料文の文中に質問文に対応する正解があるか識別する。
RTE	分類	二文の含意関係を推定する。
WNLI	分類	代名詞が置換された文が、対となる文に含まれているか識別する。
AX	分類	検証・分析用データセット。ラベルの形式は MNLI のものに近い。

第 4 章

提案手法

2.3 節において、word2vec などの出力が「単語の埋め込み表現はその単語が持つ意味を表現している」という観点に基づくと述べた。この観点をクラスタリングに適用することで、「各クラスタは当該クラスタに属する埋め込み表現群によって表現できる」という観点が得られる。本章ではこれを利用し、ラベルが付与された文を集めたデータセットを用いた事前学習済み BERT モデルの評価手法を提案する。

この提案手法は評価対象となるモデル m 及びラベルが付与された文の集合で構成されるデータセットを必要とする。その概要は以下の通りである。

1. モデル m に文を入力し、その出力から [CLS] トークンを抽出する。
2. 各 [CLS] トークンに対応する文に付与されたラベルを確認し、クラスタを構築する。
3. 各クラスタの重心を導出し、クラス内分散 A_m を計算する。
4. 全クラスタの重心の平均及びクラス外分散 B_m を計算する。
5. A_m を B_m で割った値がモデル m の評価値である。

まずモデル m に文を入力し、その出力を得る。この手法では出力全体のうち [CLS] トークンの埋め込み表現を文ベクトルとして用いる。次に、各文ベクトルに対応する文のラベルを確認し、対応するクラスタへ振り分ける。その後各クラスタの重心 $g_i^{(m)}$ を計算し、以下の式に従いクラス内分散 A_m を導出する。ただし $\sigma_i^2 = \sum_{j \in C_i} ||g_i^{(m)} - x_{i,j}||^2$

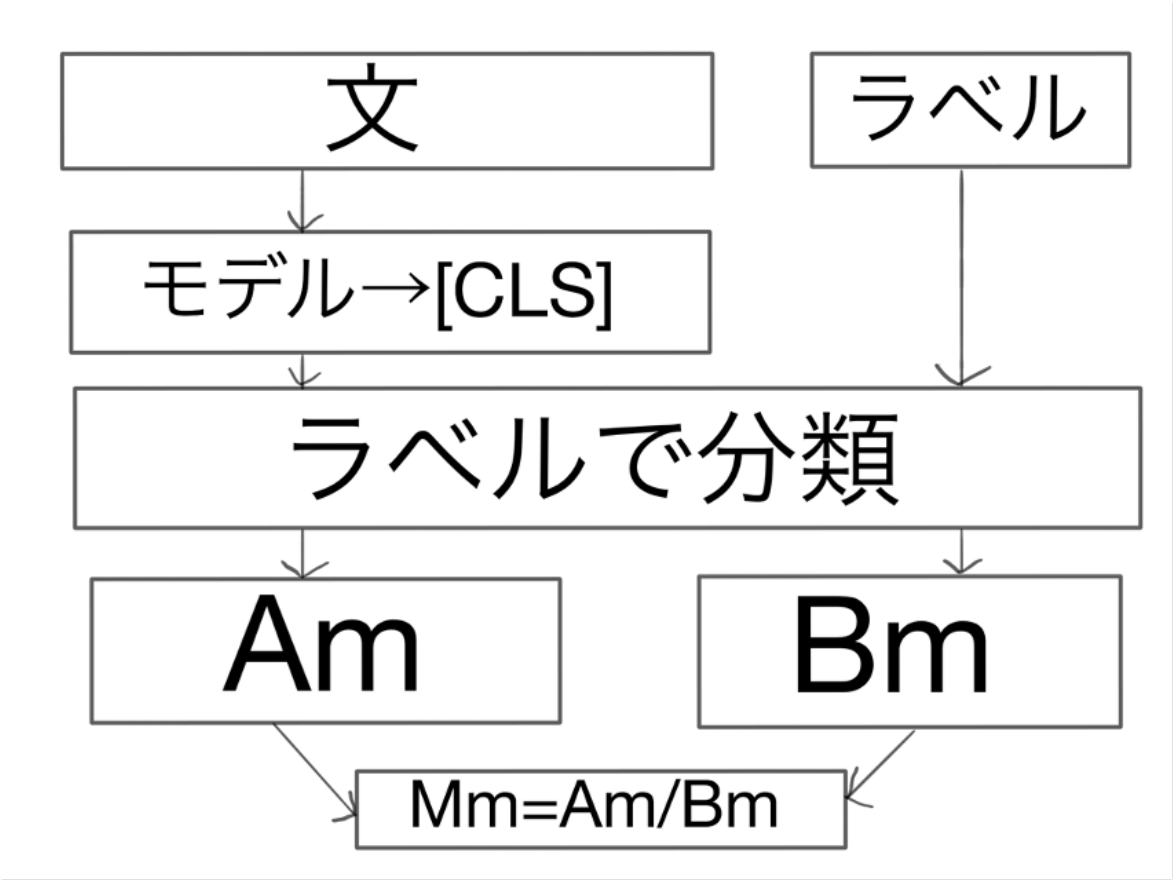

図 4.1: 提案手法の流れ

であり、C_i、N はそれぞれクラスタ i、クラスタの数である。[*1]

$$A_m = \sum_{i=1}^{N} \sigma_i^2 \tag{4.1}$$

そして全クラスタの重心の平均 $g^{(m)}$ を導出後、クラス外分散 B_m を計算することで評価の準備が整う。[*2]

$$B_m = \sum_{i=1}^{N} ||g^{(m)} - g_i^{(m)}||^2 (N = \text{クラスタの個数}) \tag{4.2}$$

モデル m の評価値を $M = \frac{A_m}{B_m}$ とする。これはモデル m の出力が適切にクラスタリングできるベクトル群であるほど小さくなる。

これまでに示してきた提案手法の流れを図 4.1 に示す。

[*1] 計算の簡略化のために偏差の二乗を分散とみなしている。
[*2] A_m の場合と同様に、偏差の二乗を分散とみなしている。

第 5 章

実験

本章では提案手法による事前学習済み日本語 BERT モデルの評価を行い、その結果を提示する。また、空所推定タスクによる評価結果を同様に提示する。

5.1 手順・設定

本論文で評価対象とする事前学習済み日本語 BERT モデルを表 5.1 に示す。各 BERT モデルの主な違いは事前学習に使用されたコーパス及び Tokenizer である。また、モデルのサイズは一部のハイパーパラメータに依存するが、本論文で評価対象とするモデルは公式 BERT [1] の Base サイズか Large サイズのいずれかに準拠している。[*1]

これ以降、各モデルを表 5.1 に掲載されている順に京大 Base(Large) 版[*2]、MeCab 版[*3]、SP 版[*4]、東北版[*5]、NICT 版[*6]、Laboro 版[*7]と記載する。

[*1] Base サイズのモデルは Attension Head の数が 12 であり、(隠れ層の) 埋め込み表現が 768 次元である 12 層構造のモデルである。これを「12-layer, 768-hidden, 12-heads」と表記する。この記法に従えば Large サイズのものは 24-layer, 1024-hidden, 16-heads と表現される。

[*2] `http://nlp.ist.i.kyoto-u.ac.jp/index.php?BERT` 日本語 `Pretrained` モデル 本研究では同サイト内で公開されているもののうち、Base 通常版 (Large WWM 版) を使用した。

[*3] `https://qiita.com/mkt3/items/3c1278339ff1bcc0187f`

[*4] `https://github.com/yoheikikuta/bert-japanese`

[*5] `https://github.com/cl-tohoku/bert-japanese`

[*6] `https://alaginrc.nict.go.jp/nict-bert/index.html` 本研究では BPE あり版を使用した。

[*7] `https://laboro.ai/column/laboro-bert/`

表 5.1: 評価対象となる事前学習済み BERT モデル

モデル公開元	サイズ	Tokenizer (特徴)	事前学習コーパス
京都大学	Base	Juman++	Wikipedia
京都大学	Large	Juman++	Wikipedia
森長	Base	MeCab + NEologd (Subword tokenize なし)	ビジネスニュース記事
Yohei Kikuta	Base	SentencePiece (do_lower_case = True)	Wikipedia
東北大学	Base	MeCab + NEologd	Wikipedia
NICT	Base	MeCab + Jumandic	Wikipedia
株式会社 Laboro	Base	SentencePiece	インターネット上のテキスト (12GB)

5.1.1　提案手法による評価

提案手法を用いたモデルの評価には Livedoor ニュースコーパス[*8]を使用する。このコーパスはデータとして、以下の 9 つのカテゴリに分類されたニュース記事を内包している。

- トピックニュース
- Sports Watch
- IT ライフハック
- 家電チャンネル
- MOVIE ENTER
- 独女通信
- エスマックス
- livedoor HOMME

[*8] `http://www.rondhuit.com/download.html#ldcc`

- Peachy

本研究ではカテゴリごとに 100 件の記事を抽出し、そのタイトルを入力文、属するカテゴリをクラスラベルとして使用し、提案手法での評価を行った。

5.1.2　空所推定タスクを用いた評価

空所推定タスクを用いた評価には Webis-CLS-10 [9] の日本語ドメインのテストデータから構築したデータセットを使用した。構築手順は次の通りである。また、この一連の流れを図 5.1 に示す。

1. ソースとなる「文を含むデータセット」全体から頻出度の高い名詞を抽出する。
2. データセット内の「抽出された名詞を含む文」を各名詞毎に一定数抽出する。
3. 抽出された名詞を対応する文のラベルとする。使用時にはラベルにした名詞を空所にする。

本研究では Webis-CLS-10 の日本語ドメインの各領域のテストデータに対し、頻出名詞 20 語を含む各 5 文、全 100 文の空所推定データセットを構築した。各領域の頻出名詞を以下に示す。

books: 本, 人, 著者, 内容, 自分, 作品, 本書, 感じ, 文章, 主人公, 小説, 部分, 最後, 言葉, 読者, 作者, 人間, 物語, 他, 世界
dvd: 映画, 作品, 人, シーン, 映像, 原作, 自分, ストーリー, 内容, ファン, 感じ, 主人公, 最後, アニメ, ドラマ, 物語, 人間, 世界, 子供, 部分
music: 曲, アルバム, 作品, 人, 音楽, 感じ, ファン, 音, バンド, 自分, 歌詞, 声, ギター, 歌, ＣＤ, 楽曲, サウンド, ライブ, シングル, 前作

ファインチューニング前の評価対象モデルを用いて、構築したデータセットに対する「空所がラベルに記載された単語である確率」を測定した。モデルの評価にはこの確率の平均値を用いた。

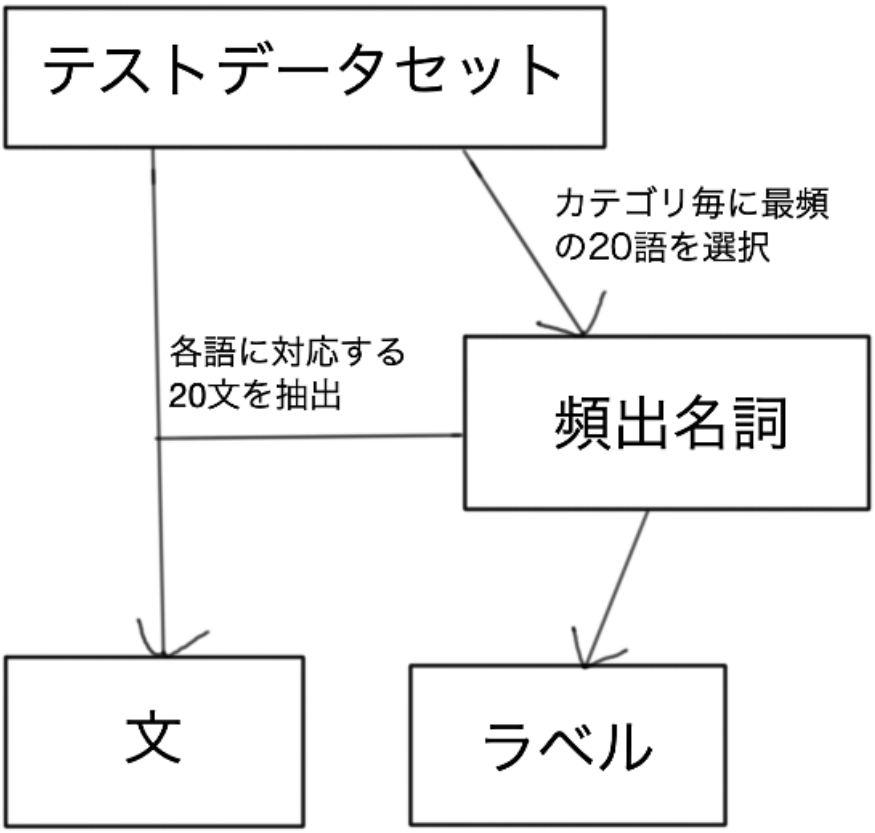

図 5.1: 文の集合から空所推定データセットを構築する方法

5.2　実験結果

まず提案手法での評価結果を表 5.2 及び図 5.2 に示す。空所推定タスクを用いた評価結果は表 5.3 及び図 5.3 に示してある。

表 5.2: 提案手法を用いたモデルの評価値

モデル	A_m	B_m	評価値
京大 Base	240131.79	337.83	710.81
京大 Large	311331.96	409.40	760.46
MeCab	97536.21	154.37	631.06
SP	67744.36	104.05	651.06
東北	49991.31	65.64	761.58
NICT	106698.11	151.27	705.37
Laboro	153378.22	273.83	**560.13**

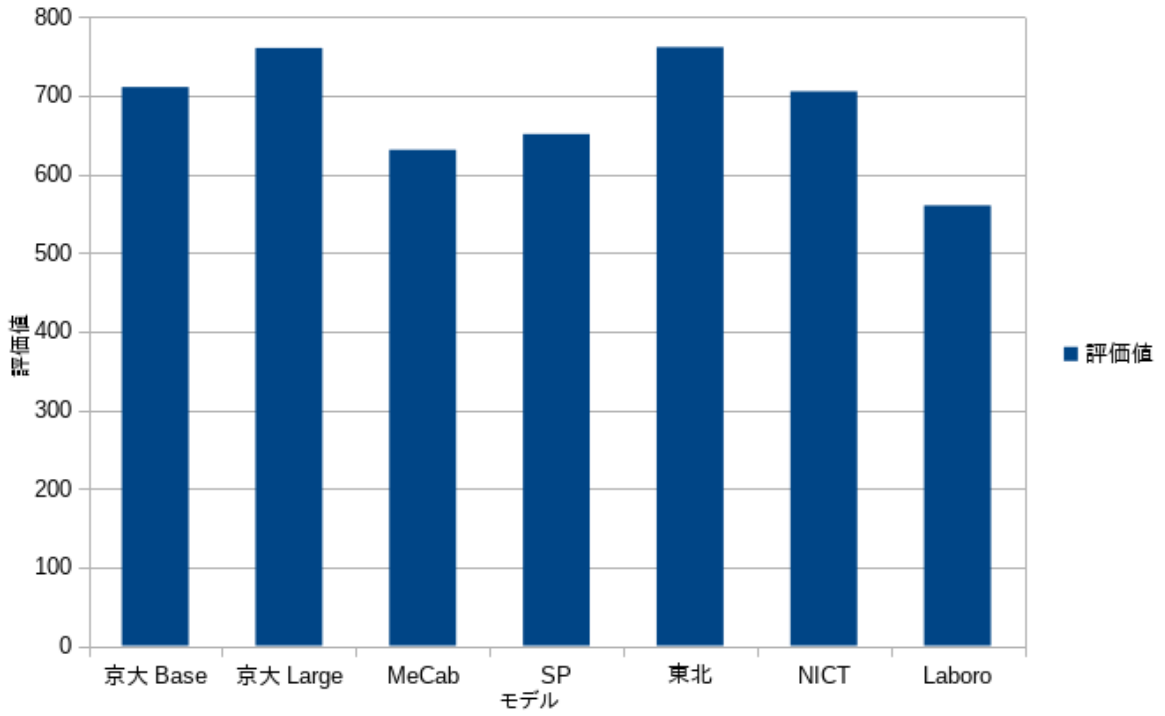

図 5.2: 提案手法を用いた各モデルの評価値

これら 2 つの表から、提案手法での評価結果は空所推定タスクを用いた評価結果とは異なる傾向を有していることが明らかになった。

表 5.3: 空所が正解の単語である確率（平均値）

モデル	books	dvd	music	全平均
京大 Base	11.53%	11.18%	9.24%	10.65%
京大 Large	14.78%	16.90%	11.52%	14.40%
MeCab	11.24%	13.62%	7.62%	10.83%
SP	7.36%	9.86%	6.41%	7.88%
東北	14.04%	12.76%	10.81%	12.54%
NICT	11.90%	12.63%	8.68%	11.07%
Laboro	8.86%	10.44%	9.85%	9.72%

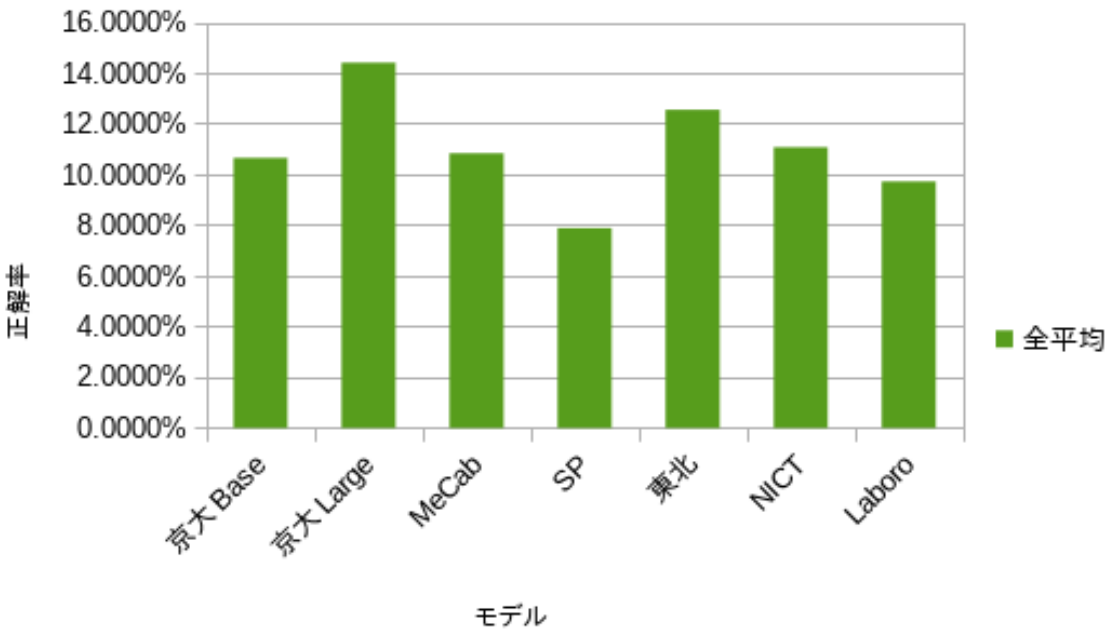

図 5.3: 空所推定タスクを用いたモデルの評価結果

第 6 章

考察

本章では 5.2 節で示した結果及び提案手法による評価結果が空所推定での評価結果と矛盾する理由について考察する。

6.1 モデルサイズの影響

まずは京都大学が公開した 2 つのモデルを用いて、モデルサイズの違いによる影響について見ていこう。表 6.1 は京大 Base 版、Large 版の評価結果を並べたものである。

表 6.1: モデルサイズの異なるモデルの評価結果

モデル	評価値 (提案手法)	全平均 (空所推定)
京大 Base	710.81	10.65%
京大 Large	760.46	14.40%

表 5.2、図 5.2 の両モデルの A_m、B_m 及び表 6.1 にも示した評価値は似た傾向を有し、京大 Base 版を定数倍したものが京大 Large 版であるかのような印象を受けた。一方、空所推定での結果は京大 Large 版のほうが優れていると示している。この結果は京大 Base 版のほうが優秀だとする提案手法での評価結果と矛盾しており、ここでも傾向の違いが確認できた。

6.2　提案手法の結果から見る各モデルの出力傾向

結果の違いについて考える前に、提案手法での評価結果を考察する。4 章の脚注で述べたように、計算の簡略化のために A_m、B_m をそれぞれクラス内分散・外分散とみなしている。そのため、A_m を比較することで同じラベルを持つ [CLS] トークン間の距離が近いかどうか判断でき、B_m で異なるラベルを持つトークン間の距離の傾向を確認できる。これらを用いることで、表 5.2 から各モデルが出力する [CLS] トークンの大まかな傾向を推測できる。

ここでは評価対象のうち、Base サイズに準拠しているもの同士を比べていこう。この評価結果によると、最も優秀なモデルは二番目に高い B_m と京大 Base 版約 100000 小さい A_m を有する Laboro 版である。二位のモデルは MeCab 版であり、東北版以上 MeCab 版未満の A_m を持つ SP 版がそれに続いた。評価値が最も低かったのが、最小の A_m 及び B_m を持つ東北版である。これはモデルが出力する全 [CLS] トークンの値が他のモデルより小さい傾向にあることを意味している。

6.3　2 つの結果間の違いについて

さて、本章冒頭で述べたように、提案手法での評価結果は空所推定での評価結果と異なる傾向を持つ。そのため、提案手法が有効であるという結論を得られなかった。なぜ傾向が違うのだろうか。

両実験にはいくつかの条件の違いがあり、完全に平等であると言うことは出来ない。これらの違いの中で最も影響が大きいだろうと思われるのが「文の種類の違い」である。提案手法での評価にはニュース記事のタイトルを使用したが、空所推定での評価で用いたのはプロダクトレビュー内の一文である。この違いが提案手法での評価結果である表 5.2 と空所推定での評価結果である表 5.3 の原因である可能性がある。

第 7 章

結論

本研究では事前学習済み日本語 BERT モデルを、ラベルを付与した文の集合及びそれらをモデルに入力した際の出力を用いて評価した。そして、以下に示す結果を得た。

Laboro 版 < MeCab 版 < SP 版 < NICT 版 < 京大 Base 版 < 東北版、　京大 Base 版 < 京大 Large 版

また、レビュー文書の分類を行う感情分析のデータセットから空所推定のデータセットを構築し、タスクベースでの評価を行った。しかし、得られた結果は以下に示す通りであり、提案手法での評価結果とは異なる傾向になった。

東北版 > NICT 版 > MeCab 版 > 京大 Base 版 > Laboro 版 > SP 版、　京大 Large 版 > 京大 Base 版

よって、本論文では提案手法の有効性を確認できなかった。しかし、今回行った 2 つの実験では使用したデータの「種別」や領域など、いくつかの違いがある。これらの違いを減らすことができれば、本論文のものとは異なる比較結果が得られる可能性がある。

謝辞

本研究を進めるにあたり、多くのご指導、ご協力を頂いた指導教員の新納浩幸教授に感謝致します。また、日々の研究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様に感謝します。

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. In *NAACL-2019*, pp. 4171–4186, 2019.

[2] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[3] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018.

[4] 芝山直希, 曹鋭, 白静, 馬ブン, 新納浩幸. 文のクラスタリングを用いた BERT 事前学習モデルの評価. 言語処理学会第 26 回年次大会, pp. 1233–1236, 2020.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, Vol. 30, pp. 5998–6008, 2017.

[6] 堺澤勇也, 小町守. 日本語動詞・形容詞類似度データセットの構築. 言語処理学会第 22 回年次大会, pp. 258–261, 2016.

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS-2013*, pp. 3111–3119, 2013.

[8] 芝山直希, 曹鋭, 白静, 馬ブン, 新納浩幸. 日本語 Pretrained BERT モデルの比較. 研究報告ドキュメントコミュニケーション (DC), Vol. 2019, No. 20, pp. 1–4, 2019.

[9] Peter Prettenhofer and Benno Stein. Cross-Language Text Classification using Structural Correspondence Learning. In *48th Annual Meeting of the Association of Computational Linguistics (ACL 10)*, pp. 1118–1127. Association for Compu-

tational Linguistics, July 2010.

付録

A　プログラムリスト

本章では、本論文のための研究に使用したプログラムを示す。なお、一部のプログラムは Jupyter Notebook 形式 (.ipynb)[*1]にて記述したが、当該形式は JSON の書式に従って記述されているため、対応アプリケーション以外での可読性が低い。その為、本章では Python のソースコードの形式 (.py) に変換したものを掲載する。

A.1　提案手法での実験用プログラム

本節では、提案手法による評価のために作成・使用したプログラムを示す。本論文では livedoor ニュースコーパスから記事タイトルを抽出する工程にプログラム A.1 を、各モデルに抽出したデータセットを入力し [CLS] トークンを得る工程にプログラム A.2 を Jupyter Notebook 形式で記述したもの (これ以降は単に Notebook と表記する) を、そして獲得したトークンと所属するカテゴリの情報から評価値を計算する工程にプログラム A.3 を使用した。

ソースコード A.1: タイトル抽出プログラム extract-title.py

```
1 #!/usr/bin/env python3
2
3
4 __doc__ = """extract title from news text.
5
6 Usage:
7     extract-title.py <src_dir> [-h|--help] [--title_line=<n_line>][--
          output=<out_dir>]
8     extract-title.py gen_ssv_dataset <src_dir>... [-h|--help] [-l|--
          length=<n_len>|--n_test=<num_test>|--n_validate=<num_val>]
```

[*1] Python の対話型実行環境の機能を持つ Web アプリケーション。実行可能なプログラムを埋め込んだ文書を作成できる。コードを任意数のブロックに分割して記述し、ブロック毎にコードの編集・実行ができるという特徴を持つ。

```
9
10 Options:
11     --output=<out_dir> where to save results [default: ./result/]
12     --title_line=<n_line> line number of title [default: 3]
13     -l --length=<n_len> nums of whole dataset(dataset mode) [default:
           240]
14     --n_test=<num_test> nums of test data(dataset mode) [default: 20]
15     --n_validate=<num_val> nums of validation data(dataset mode) [default
           : 20]
16     -h, --help show this message
17 """
18
19 from docopt import docopt
20 from pathlib import Path
21 import random
22
23 def sample_titles(titles, n_target, seed=20200122):
24     random.seed(seed)
25     return random.sample(titles, n_target)
26
27 def extract_all_titles(filepath, title_line):
28     texts = filepath.glob("*.txt")
29     whole_text = []
30     for txtfile in texts:
31         whole_text.append(extract_title(txtfile, title_line))
32     return whole_text
33
34 def extract_title(filepath, target_line):
35     with open(filepath, "r") as f:
36         for i, text in zip(range(target_line), f):
37             if i+1 == target_line:
38                 return text
39
40 def write_ssv(filepath, ssv_dataset):
41     ssv_text = []
42     with open(filepath, "w") as f:
43         for i in ssv_dataset:
44             label, title = i[0], i[1]
45             ssv_text.append('␣'.join([str(label), title]))
46         f.writelines(ssv_text)
47
48 args = docopt(__doc__)
49 out_d = Path(args['--output'])
50 title_line = int(args['--title_line'])
51 print("extracting...")
52
53 if not out_d.is_dir():
54     out_d.mkdir()
55 if args['gen_ssv_dataset']:
56     src = [Path(d) for d in args['<src_dir>']]
57     n_label = len(src)
58     len_dataset = int(args['--length'][0])
59     n_test_data = int(args['--n_test'])
60     n_val_data = int(args['--n_validate'])
61     n_train_data = len_dataset - n_test_data - n_val_data
62     train_dataset = []
63     test_dataset = []
64     validate_dataset = []
65     for i, root in zip(range(n_label), src):
66         titles = extract_all_titles(root, title_line)
67         sampled = sample_titles(titles, len_dataset)
```

```
68          data = [(i, x) for x in sampled]
69          train_dataset += data[:n_train_data]
70          test_dataset += data[n_train_data:n_train_data + n_test_data]
71          validate_dataset +=data[n_train_data + n_test_data:]
72       write_ssv(out_d.joinpath("train.ssv"), train_dataset)
73       write_ssv(out_d.joinpath("test.ssv"), test_dataset)
74       write_ssv(out_d.joinpath("validation.ssv"), validate_dataset)
75
76    else:
77       src = Path(args['<src_dir>'])
78       with out_d.joinpath(src.name+"-titles.txt").open(mode="w") as result:
79          for src_file in texts:
80             t = extract_title(src_file, linenumber)
81             result.write(t)
```

ソースコード A.2: ベクトル取得プログラム extract-ndarray.py

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  import torch
8  from transformers import BertConfig, BertTokenizer, BertModel,
       BertJapaneseTokenizer
9  from mytokenizers import JumanTokenizer, JumanPreprocess, JumanBertEncode
       , SentencePieceTokenizer
10 import numpy as np
11 import gc
12 from tqdm.notebook import tqdm
13 from pathlib import Path
14
15
16 # In[2]:
17
18
19 #ベースディレクトリ、分かち書きファイル、抽出単語などの各種設定を指定
20 BASEDIR = Path("./src/livedoor-news/")
21 BASENAME = ""
22 TARGET = BASEDIR.joinpath(BASENAME)
23 target_word = "[CLS]"
24 BERTBASE = "./jumanpp/BERT-large-wwm/"
25 BERTDIR = Path(BERTBASE+"model")
26 use_gpu = True
27 use_preset = False
28 tokenizer_type = "jumanpp"
29 mecab_dic_dir = "/usr/lib/mecab/dic/mecab-ipadic-neologd/"
30 sentencepiece_model = str(BERTDIR.joinpath("wiki-ja.model"))
31 sentencepiece_vocab = sentencepiece_model.rstrip(".model")+".vocab"
32 use_subword_tokenize = True if "neologd" in mecab_dic_dir else False
33
34
35 # In[3]:
36
37
38 if use_preset:
39    conf = BertConfig.from_pretrained("bert-base-japanese")
40    bertmodel = BertModel.from_pretrained("bert-base-japanese")
41 else:
```

```
42     conf = BertConfig.from_json_file(BERTDIR.joinpath("config.json"))
43     bertmodel = BertModel.from_pretrained(BERTDIR.joinpath("pytorch_model.
           bin"), config=conf)
44 bertmodel.to("cuda") if use_gpu else bertmodel.to("cpu")
45 bertmodel.eval()
46 texts = TARGET.glob("*.txt")
47
48
49 # In[4]:
50
51
52 if use_preset:
53     tokenizer = BertJapaneseTokenizer.from_pretrained("bert-base-japanese")
54 elif tokenizer_type == "jumanpp":
55     tokenizer = JumanTokenizer.from_pretrained(str(BERTDIR))
56 elif "mecab" in tokenizer_type:
57     tokenizer = BertJapaneseTokenizer.from_pretrained(str(BERTDIR),
           word_tokenizer_type="mecab", mecab_option="--dicdir="+mecab_dic_dir,
            do_subword_tokenize=use_subword_tokenize)
58 elif tokenizer_type == "sentencepiece":
59     tokenizer = SentencePieceTokenizer(sentencepiece_model,
           sentencepiece_vocab)
60
61 else:
62     raise ValueError("tokenizer_type␣is␣unseted")
63
64
65 # In[5]:
66
67
68 for text in tqdm(texts):
69     tokens = []
70     with open(text, "r") as t:
71        for line in t:
72            if tokenizer_type in ["sentencepiece"]:
73                token = tokenizer.tokenize(line.rstrip("\n"))
74                token = tokenizer.convert_tokens_to_ids(["[CLS]"]+token+["[SEP
                       ]"])
75            else:
76                token = tokenizer.encode(line.rstrip("\n"))
77            sentence = tokenizer.convert_ids_to_tokens(token)
78            if target_word in sentence:
79                tdx = sentence.index(target_word)
80            elif target_word == "[CLS]":
81                tdx = 0
82            with torch.no_grad():
83                inputs = torch.tensor([token]).to("cuda") if use_gpu else
                       torch.tensor([token])
84                generated = bertmodel(inputs, token_type_ids=None)[0]
85            data = generated[0][tdx].to("cpu").tolist() if use_gpu else
                    generated[0][tdx].tolist()
86            tokens.append(data)
87     dataarray = np.array(tokens)
88     print(dataarray.shape)
89     np.savez_compressed(text.parent.joinpath(str(text.name).rstrip(".txt")+
           ".npz"), dataarray)
90
91
92 # In[6]:
93
94
```

```
95 print("arrays␣generated!")
```

ソースコード A.3: 評価値計算用プログラム evaluate-clustering.py

```
1  """Evaluate output with clustering.
2
3  usage: evaluate_clustering.py [-h] [--cluster_length=<N>] [-q|--quiet] <
       model_root_dir>...
4
5  options:
6     --cluster_length=<N> how many datas a cluster has [default: 100]
7     -q, --quiet print only root score
8     -h, --help show this message
9  """
10
11 from docopt import docopt
12 from pathlib import Path
13 import numpy as np
14
15 def text2path(path_str):
16    return Path(path_str)
17
18 #
19 def evaluate_model(root_dir,cluster_length):
20    root = []
21    dirs = list(root_dir.glob("*.npz"))
22    for cluster in dirs:
23       data = np.load(cluster, allow_pickle=True)
24       root.append(data["arr_0"])
25    root = np.concatenate(root)
26    gi = []
27    print(root.shape)
28    Am_i = np.zeros((len(dirs), root.shape[1]))
29    for i in range(0,len(root),cluster_length):
30       cluster = root[i:i+100]
31       g_i = np.sum(cluster,axis=0) / cluster_length
32       for div in g_i - cluster:
33          Am_i[int(i/cluster_length)] += np.abs(div)**2
34       gi.append(g_i)
35    g = np.sum(gi,axis=0) / len(gi)
36    Am = np.sum(Am_i)
37    Bm_i = np.abs(gi - g)**2
38    Bm = Bm_i.sum()
39    return Am/Bm, Am, Bm
40
41 if __name__ == '__main__':
42    args = docopt(__doc__)
43    root_dirs = map(text2path, args['<model_root_dir>'])
44    root_dirs = list(root_dirs)
45    cluster_length = int(args['--cluster_length'])
46    for d, i in zip(root_dirs, range(len(root_dirs))):
47       M, Am, Bm = evaluate_model(d, cluster_length)
48       print("Model"+str(i)+"␣Score:"+str(M))
49       if not args['--quiet']:
50          print("Am:"+str(Am))
51          print("Bm:"+str(Bm))
```

A.2 空所推定タスクでの評価用プログラム

本節では、空所推定タスクを用いた評価のために作成・使用したプログラムを示す。本論文上において、各モデルに空所推定タスクを遂行させ正解単語である確率を取得し、その平均値を計算する Notebook を作成した。これを Python のプログラムに変換したものをプログラム A.4 に示す。

ソースコード A.4: eval-masked-prediction.py

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import torch
import random
import json
from transformers import BertConfig, BertTokenizer, BertForMaskedLM,
    BertJapaneseTokenizer
from mytokenizers import JumanTokenizer, SentencePieceTokenizer
from torch.nn.functional import softmax
from tqdm.notebook import tqdm
from pathlib import Path


# In[2]:


#ベースディレクトリ、分かち書きファイル、スライス設定とを指定 JSON
BASEDIR = Path("./src/cls-amazon/fill-mask/")
BASENAME = ""
TARGET = BASEDIR.joinpath(BASENAME)
srcfile_filter = "*"
src_type = ".dat"
max_seq_length = 512
BERTBASE = "./jumanpp/BERT-large-wwm/"
BERTDIR = Path(BERTBASE+"model")
use_preset = False
use_gpu = True
tokenizer_type = "jumanpp"
mecab_dic = "/usr/lib/mecab/dic/mecab-ipadic-neologd"
sentencepiece_model = str(BERTDIR.joinpath("webcorpus.model"))
sentencepiece_vocab = str(sentencepiece_model).rstrip(".model")+".vocab"
random_seed = 20200424
mask_sentence_sep = "␣"
retokenize = True
pre_tokenized = True
lower_case = False
use_subword = True if "jumandic" in tokenizer_type else False


# In[3]:


```

```
46 if use_preset:
47     conf = BertConfig.from_pretrained("bert-base-japanese")
48     bertmodel = BertForMaskedLM.from_pretrained("bert-base-japanese")
49 else:
50     load_ckpt = not BERTDIR.joinpath("pytorch_model.bin").exists()
51     conf = BertConfig.from_json_file(BERTDIR.joinpath("config.json"))
52     bertmodel = BertForMaskedLM.from_pretrained(BERTDIR, config=conf,
           from_tf=load_ckpt)
53 bertmodel.eval() if not use_gpu else bertmodel.to("cuda").eval()
54 texts = TARGET.glob(srcfile_filter+src_type)
55
56
57 # In[4]:
58
59
60 if use_preset:
61     tokenizer = BertJapaneseTokenizer.from_pretrained("bert-base-japanese")
62 elif "mecab" in tokenizer_type:
63     tokenizer = BertJapaneseTokenizer.from_pretrained(str(BERTDIR),
           do_subword_tokenize=use_subword, word_tokenizer_type="mecab",
           mecab_option="--dicdir="+mecab_dic)
64 elif tokenizer_type == "sentencepiece":
65     tokenizer = SentencePieceTokenizer(sentencepiece_model,
           sentencepiece_vocab, do_lower_case=lower_case)
66 elif tokenizer_type == "jumanpp":
67     tokenizer = JumanTokenizer.from_pretrained(str(BERTDIR))
68 else:
69     raise ValueError("tokenizer_type␣or␣use_preset␣flag␣is␣unseted")
70 if tokenizer_type == "sentencepiece":
71     mask_token = "[MASK]"
72     cls_token = "[CLS]"
73     sep_token = "[SEP]"
74 else:
75     mask_token = tokenizer.mask_token
76     cls_token = tokenizer.cls_token
77     sep_token = tokenizer.sep_token
78
79
80 # In[5]:
81
82
83 def tokenize_sentence(tokenize_func, sentence):
84     global cls_token, sep_token
85     sentence = tokenize_func(sentence.rstrip("\n"))
86     if len(sentence) > max_seq_length-2 and sentence[0] != cls_token:
87         sentence = [cls_token]+sentence[:max_seq_length-2]+[sep_token]
88     elif len(sentence) > max_seq_length-1 and sentence[0] == cls_token:
89         sentence = sentence[:max_seq_length-1]+[sep_token]
90     elif sentence[0] != cls_token:
91         sentence = [cls_token]+sentence+[sep_token]
92     return sentence
93
94
95 # In[6]:
96
97
98 n_sentence = []
99 p_true = []
100 for text in tqdm(list(texts)):
101     predict_result = []
102     random.seed(random_seed)
```

```
103     p_correct = 0
104     n_mask = 0
105     with open(text, "r") as t:
106        for line in t:
107           if mask_sentence_sep != None:
108              words = line.split(mask_sentence_sep)
109              masked_word = words[0].lower() if lower_case else words[0]
110              context = words[1:]
111              if retokenize and pre_tokenized:
112                 l = "".join(context).replace(masked_word, mask_token, 1)
                        if tokenizer_type not in ["sentencepiece", "jumanpp"]
                        else "".join(context)
113                 if tokenizer_type == "jumanpp":
114                    sentence = tokenizer.juman_tokenize(l.rstrip("\n"))
115                 else:
116                    sentence = tokenize_sentence(tokenizer.tokenize, l)
117              elif retokenize:
118                 l = context.replace(masked_word, mask_token) if
                        tokenizer_type not in ["sentencepiece", "jumanpp"] else
                         context
119                 if tokenizer_type == "jumanpp":
120                    sentence = tokenizer.juman_tokenize(l.rstrip("\n"))
121                 else:
122                    sentence = tokenize_sentence(tokenizer, l)
123              else:
124                 sentence = context
125                 sentence[context.index(masked_word)] = mask_token
126              if retokenize and tokenizer_type in ["sentencepiece", "jumanpp"
                    ]:
127                 for i, token in enumerate(sentence):
128                    if masked_word in token:
129                       splited = sentence[i].replace(masked_word, "␣"+
                             mask_token+"␣").split("␣")
130                       sentence[i:i+1] = [t for t in splited if t != ""]
131                       break
132                    elif [c for c in masked_word] in sentence[i:]:
133                       endidx = sentence.index(masked_word[-1])
134                       sentence[i:endidx+1] = [mask_token]
135                       break
136              if tokenizer_type == "jumanpp":
137                 sentence = tokenize_sentence(tokenizer.bert_tokenize, "␣"
                        .join(sentence))
138              masked_idx = sentence.index(mask_token)
139           else:
140              sentence = tokenize_sentence(tokenizer.tokenize, line.rstrip(
                    "\n"))
141              masked_idx = random.randrange(1, len(sentence)-1)
142              masked_word = sentence[masked_idx]
143           sentence[masked_idx] = mask_token
144           masked_id = tokenizer.convert_tokens_to_ids([masked_word])[0] if
                 tokenizer_type == "sentencepiece" else tokenizer.
                convert_tokens_to_ids(masked_word)
145           token = tokenizer.convert_tokens_to_ids(sentence)
146           with torch.no_grad():
147              input_tensor = torch.tensor([token]) if not use_gpu else
                    torch.tensor([token]).to("cuda")
148              generated = bertmodel(input_tensor)[0]
149           prediction_logits = softmax(generated[0, masked_idx, :], dim=0).
                to("cpu") if use_gpu else softmax(generated[0, masked_idx,
                :], dim=0)
150           predicted_token_logit = prediction_logits[masked_id]
```

```
151             predicted = tokenizer.convert_ids_to_tokens([torch.argmax(
                    prediction_logits)])[0]
152             p_correct += predicted_token_logit
153             n_mask += 1
154             predict_result.append({"masked_sentence": "".join(sentence).
                    replace("##", ""), "predicted": predicted, "masked_word":
                    masked_word, "masked_word_percentage": predicted_token_logit.
                    tolist()})
155     print("accuracy␣({1}):{0:.4f}".format(float(p_correct / n_mask), text.
            name))
156     p_true.append(p_correct)
157     n_sentence.append(n_mask)
158     with open(str(text).rstrip(src_type)+".json", "w") as r:
159         json.dump(predict_result, r)
160
161
162 # In[7]:
163
164
165 result = float(sum(p_true) / sum(n_sentence))
166 print("model␣accuracy:␣{}".format(result))
```