令和２年度 茨城大学大学院理工学研究科情報工学専攻
修士論文

２言語 BERT を利用したターゲット言語の教師データを必要としない感情分析

令和３年２月５日

情報工学専攻
19NM716G 荘司 響之介
新納 浩幸 教授

令和 3 年茨城大学大学院理工学研究科情報工学専攻修士論文

2 言語 BERT を利用した
ターゲット言語の教師データを必要としない感情分析

19NM716G 荘司 響之介
新納 浩幸 教授

論文要旨

本論文では、英語のようなメジャー言語を用いて学習した判別機を利用してマイナー言語の文書の感情分析を行うことを目的とし、その実現のために Billingual Word Embeddings (以下、BWE) と BERT を用いた。BWE とは二言語間の分散表現を同一のベクトル空間上に表現する技術である。これにより、ある言語で学習したモデルをそのまま別言語のタスクに利用することを試みた。

感情分析とはレビュー文書が肯定的なものか、否定的なものかを判定するタスクである。これは文書分類の一種であり、教師あり学習を用いて解決できる。しかし教師あり学習には大量のラベル付きデータ（教師データ）が必要であり、このデータの構築コストが高いという問題がある。ただし英語などのメジャーな言語に対しては、ラベル付けされたデータが既に存在していることも多い。この場合、英語側では分類器を学習できるため、その学習できた知識を、タスクの対象となっている言語側へ転移できれば、ターゲット言語での教師データを利用せずに、分類器を構築できる。本論文はそのような転移を行うために BWE を利用する。

本論文では英語のラベル付き文書を BERT を用いてベクトル化し、そのベクトルを基に分類器を学習する。次にターゲット領域の文書となる日本語文書を BERT を用いてベクトル化し、BWE の原理を用いて変換した後先の分類器によって識別する。これによってターゲット領域側のラベル付き文書を全く利用せずに感情分析が可能となる。

実験では、提案手法（BWE と BERT を用いて日本語の教師データを用いずに日本語の感情分析を行う手法）と「BWE と Word2Vec を用いて日本語の教師データを用いずに日本語の感情分析を行う手法」を比較した。その結果前者の方が、「テスト文書が英語の場合」から「テスト文書が日本語の場合」への精度の低下が少かった。Word2Vec が単語そのものの意味をベクトル化するのに対し、BERT では文脈に沿った単語の意味をベクトル化することが原因だと考えられる。

提案手法では「テスト文書が英語の場合」の精度が低かった。今後、その精度を向上させることが出来ればそれに伴って「テスト文書が日本語の場合」の精度もよくなると考えられる。今後はその方向で研究を進めたい。

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

Sentiment analysis that does not require training data in the target language using billingual BERT

19NM716G Kyounosuke syouji
Hiroyuki Shinnou

ABSTRACT

In this paper, attenpt sentiment analysis without using training data by using BERT and Bilingual Word Embeddings (BWE).BWE is a technology that expresses the distributed representation between two languages on the same vector space.This will I tried to use the model learned in one language as it is for tasks in another language.

Sentiment analysis is the task of determining whether a review document is positive or negative. This is a type of document classification that can be solved using supervised learning. But, there is a problem that large amounts of training data is required in supervised learning and the construction cost of this data is high. However, in major languages such as English, labeled data often already exists. In this case, since the classifier can be learned on the English side, if the learned knowledge can be transferred to the language side that is the target of the task, the classifier can be constructed without using the training data in the target language. In this paper, use principles of BWE to make such a transition.

In this paper, English labeled documents are vectorized using BERT. And learn a classifier based on that vector. Next, the Japanese document in target area is vectorized using BERT and After conversion using the BWE principle, it is identified by the classifier. By this way, Sentiment analysis becomes possible without using any labeled documents on the target area side.

In the experiment, the proposed method (A method for Japanese sentiment analysis using BWE and BERT without using Japanese training data) and "A method for Japanese sentiment analysis using BWE and Word2Vec without using Japanese training data" were compared. As a result, The former had less decrease in accuracy from "A case test document is in English" to "A case test document is in Japanese". The reason is that Word2Vec vectorizes the meaning of the word itself, while BERT vectorizes the meaning of the word in context.

The accuracy of "A case test document is in English" was low in the proposed method. If the accuracy can be improved in the future, the accuracy of "A case test document is in Japanese" will be improved accordingly.

目 次

表 目 次

図 目 次

第1章　序論

本論文では BERT と Bilingual Word Embeddings (以下、 BWE) を用いることで、教師データを利用しない感情分析を試みる。

感情分析とはレビュー文書が肯定的なものか、否定的なものかを判定するタスクである。これは文書分類の一種であり、教師あり学習を用いて解決できる。しかし教師あり学習には大量のラベル付きデータ（教師データ）が必要であり、このデータの構築コストが高いという問題がある。ただし英語などのメジャーな言語に対しては、ラベル付けされたデータが既に存在していることも多い。この場合、英語側では分類器を学習できるため、その学習できた知識を、タスクの対象となっている言語側へ転移できれば、ターゲット言語での教師データを利用せずに、分類器を構築できる。本論文はそのような転移を行うために BWE の原理を利用する [1]。

BWE とは英語や日本語などの異なる言語の分散表現を共通に扱う枠組みである。例えば英語 “dog” の分散表現と日本語「犬」の分散表現は、学習基のコーパスが異なるために、異なるベクトルであるが、概念としては同じなので、同一のベクトルとして表現できるはずである。このようなアイデアのもと、異なる言語の分散表現を同一したもの、あるいはそれらの変換を実現したものが BWE である。本論文ではそのような同一化を単語ベクトル単位ではなく文書ベクトル単位で行う。

提案手法としては、英語のラベル付き文書を BERT を用いてベクトル化し、そのベクトルを基に分類器を学習する。次にターゲット領域の文書となる日本語文書を BERT を用いてベクトル化し、それを上記の　BWE のように変換したものを先の分類器によって識別する。これによってターゲット領域側のラベル付き文書を全く利用せずに、感情分析が可能となる。

実験では、提案手法（日本語の教師データを用いて日本語の感情分析を行う手法）と、単純な英語のみの感情分析を比較した。

第2章　Bilingual Word Embeddings

2.1　BWEの構築方法

まず、BWEの構築方法について簡潔に述べる。 BWE の構築には４つのアプローチが用いられる。

１つ目は、モノリンガルマッピングである。このアプローチでは、単一言語の分散表現をその言語のコーパスから作成し、違う言語同士の分散表現を用いて線形変換を学習する。これにより、未知の単語をソース言語からターゲット言語にマッピングできる [2]。２つ目は、疑似クロスリンガルである。このアプローチは、まず違う言語が混在したコーパスを作成し、既存の分散表現モデルをそのコーパス上で学習するという手法である [3]。３つ目は、クロスリンガルである。パラレルコーパス上でそれぞれの分散表現を学習すると同時に、異なる言語間の制約を最適化することで、似た意味の単語の分散表現を共有空間上で近づけることができる [4]。４つ目は、ジョイントオプティマイゼーションである。このアプローチでは、異なる言語間の最適化に連帯して、単一言語の組み合わせ最適化とクロスリンガルの損失関数の最適化を行う [5]。

本論文では、１つ目のアプローチを用いて BWE を構築した。

2.2　モノリンガルマッピングによるBWEの構築

2.2.1　線形プロジェクション

Mikolov ら [2] はベクトル空間は単語間の意味関係を記号化することができるという概念を流行らせた。さらに彼らは、単語間の幾何学的関係は言語を通じて似ているということに気づいた。その結果、変換行列 W を用いた線形変換を活用することにより、ある言語のベクトル空間を別の言語のベクトル空間に変換できる可能性が示唆された。

まず、約 5000 語のソース言語の頻出単語をターゲット言語に翻訳しそれをバイリンガル辞書とする。そして、変換行列 W によって変換されたソース単語 w_i のベクトル表現 x_i とその翻訳語のベクトル表現 z_i の距離を最小にすることによって、確率的勾配下降法で変換行列 W を学習する。

$$\min_W \sum_{i=1}^{n} |Wx_i - z_i|^2$$

2.2.2 正規化と直行変換

Xing ら [6] は Mikolov ら (2013) による線形プロジェクションの矛盾に気づきその解決に着手した。Mikolov らは最初に単一言語の分散表現を学習していたことを思い出してほしい。その際、彼らは以下のような skip-gram モデルの目的関数を使用している。

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{-C\leq j\leq C, j\neq 0}\log P(w_{i+j}|w_i)$$

ここで、C はコンテキストサイズを表している。また、$P(w_{i+j}|w_i)$ は softmax を使って計算される。

$$P(w_{i+j}|w_i)=\frac{\exp(c_{w_{i+j}}^T c_{w_i})}{\sum_w \exp(c_w^T c_{w_i})}$$

そして、２つの単一言語のベクトル空間同士の線形変換を学習する。

$$\min_W \sum_{i=1}^{n}|Wx_i - z_i|^2$$

Xing らは、単語表現を学習する目的関数（内積を基にした最尤法）、単語空間の距離尺度（コサイン類似度）、線形変換を学習する目的関数（平均二乗誤差）が不適合であるためパフォーマンスが低下する可能性があると論じた。

訓練中に使われる内積による類似性の尺度 $c_w^T c'_w$ とテストのために使われるコサインによる類似性の尺度 $\frac{c_w^T c_{w'}}{|c_w||c_{w'}|}$ の間の不具合を改善するために、内積をテストで使用することができるだろう。

しかし、コサイン類似度は NLP の評価尺度としてとても便利であり、一般的に内積よりもパフォーマンスがよい。そのため、彼らは訓練中に単語ベクトルを正規化することで内積とコサイン類似度を同値する方法を提案した。その結果すべてのベクトルは図 2.1 のように球状に配置される。

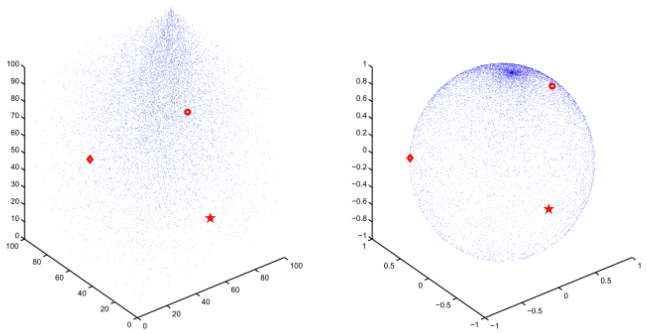

図 2.1: 正規化する前 (左) と後 (右) の単語表現

次に彼らは、変換を学習する際にコサイン類似度を使用することで、コサイン類似度の尺度と最小二乗誤差の間の矛盾を解決した。

$$\max_W \sum_i (Wx_i)^T z_i$$

最後に、W に直行行列という制約を与えて投影されたベクトル Wx_i を正規化する。

2.2.3 Max-margin と intruders

Lazarridou ら [7] は Mikolov ら (2013) による線形変換の目的関数に別の問題があることに気づいた。というのも、彼らは最小二乗法を変換行列を学習するための目的関数として使用すると、hubness が起こる、すなわち一部の単語がたくさんの他の単語の最も近くにある単語として現れる傾向にあることを発見した。彼らは margin-based(max-margin) ranking loss(Collobert ら [8]) を使い、$\widehat{y}_i$ と投影されるソース単語 x_i の正しい翻訳ベクトル y_i が他のターゲット単語 y_j よりも高くランク付けされるようにすることでこの問題を解決した。

$$\sum_{j \neq i}^{k} \max\{0, \gamma + \cos(\widehat{y}_i, y_i) - \cos(\widehat{y}_i, y_j)\}$$

ここで、k は負例の数、γ はマージンである。

彼らは、最小二乗損失を超える max-margin を選択するとパフォーマンスが改善され hubness が減ることを示した。さらに、どのモデルが正しい翻訳を高くランク付けするか比べることが重要である。有益な負例は intruder (今回の例の場合は”truck”) である。図 2.2 のように、投影ベクトル $\widehat{y}_i$ の近くにあるが実際の翻訳ベクトル y_i からは遠い場所にある。

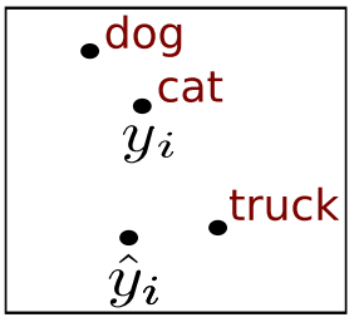

図 2.2: ”cat”の負例として”dog”ではなく intruder である”truck”が選ばれる

これらの intruders はモデルがターゲット関数の近似をかなり失敗しているケースを判別できるようにし、それにより振る舞いを訂正させる。彼らは、最急勾配法の全段階で $s_j = \cos(\widehat{y}_i, y_j) - \cos(y_i, y_j)$ を計算し最も大きい s_j を x_i の負例とした。ランダムな負例の代わりに intruders を使うことで、タスクが 2%程度改善された。

2.2.4 直交変換・正規化・平均による中心化

Artexe ら [9] は先行研究を一般化した。彼らは cross-lingual 表現の学習を改善するための様々な制約を提案した。

まず、直行制約である。マッピング後も内積を保存するためには単一言語の不変性が必要であり、単一言語のタスク (例えば word-analogy) のパフォーマンスの劣化を防ぐことができる。単一言語の不変性を得るためには直交行列 $W(W^TW = I)$ を要する。直行制約のような厳密解は以下の式で与えられる。

$$W = VU^T$$

ここで、$Z^TX = U\Sigma V^T$ は Z^TX の特異値分解である。

次に、すべての単語ベクトルが目的関数に等しく寄与するのを確実にするために正規化を行う。

$$\underset{W}{\operatorname{argmin}} \sum_i |W\frac{x_i}{|x_i|} - \frac{z_i}{|z_i|}|^2$$

$$= \underset{W}{\operatorname{argmin}} \sum_i \cos\left(Wx_i, z_i\right)$$

最後に、Artetxe らは 2 つの無作為に取り出した単語は大体似ていないと予測されるため、それらのコサイン類似度は 0 になるはずであると論じた。彼らは中心化行列 C_m を用いて次元的な平均による中心化を行った。

$$\underset{W}{\operatorname{argmin}} \sum_i \|C_mWx_i - C_mz_i\|^2$$

$$= \underset{W}{\operatorname{argmax}} \sum_i \operatorname{cov}(Wx_i, z_i)$$

2.2.5 線形変換のマルチステップフレームワークによる BWE

Artetxe ら [10] は、先行研究を一般化するようなマッピングを学習するマルチステップフレームワークを提案した。フレームワークの i 番目のステップでは、線形変換をそれ以前のステップで出力された単語ベクトルに適用する。つまり、ステップ i での単語ベクトルと変換行列をそれぞれ $X_{(i)}$、$W_{x(i)}$ とすると以下の式が成り立つ。

$$X_{(i)} = X_{(i-1)}W_{x(i)}$$
$$W_X = \prod_i W_{x(i)}$$

このフレームワークは具体的には以下のステップで成り立つ。

- Step 0: 正規化 (任意):

  このステップでは単語ベクトルに対し正規化、平均による中心化を行う。このステップは前処理的なステップで初期行列 $X_{(0)}$、$Z_{(0)}$ を得る。

- Step 1: ホワイトニング (任意):

  このステップではそれぞれの言語の単語ベクトルにホワイトニングを適用する。これにより、それらの異なる構成要素は分散 1 をもち相関せず、共分散行列は単位行列になる。そのために、Mahalanobisor ZCA whitening を採用している。

$$W_{X(1)=(X^TX)^{-\frac{1}{2}}}$$
$$W_{Z(1)=(Z^TZ)^{-\frac{1}{2}}}$$

- Step 2: 直行マッピング

  このステップではそれぞれの言語の単語ベクトルを共有空間にマッピングする。それらの変換は直行に制約され、それぞれの言語内で内積を保存する。具体的には次のように変換する。

$$W_{X(2)} = U$$
$$W_{Z(2)} = V$$

  ここで、$USV^T = X_{(1)}^T Z_{(1)}$ は $X_{(1)}^T Z_{(1)}$ の特異値分解である。

  これにより、マッピングされた単語ベクトルの累積的な共分散 $\mathrm{Tr}(X_{(1)}W_{X(2)}W_{Z(2)}^T Z_{(1)}^T)$ が最大化される。

- Step 3: 再ホワイトニング (任意)

  このステップでは相関に照らし合わせてそれぞれの成分を再ホワイトニングし、それらの関連性を増加させる。形式化が簡単になるように、ステップ 1 を適用した場合のみこのステップだけ考慮する。この場合、相関は並外れた数値 S(step2) に相当する。再ホワイトニングはソース言語に適用 $(W_{X(3)} = S, W_{Z(3)=I})$ またはターゲット言語に適用 $(W_{X(3)} = I, W_{Z(3)} = S)$ することができる。

- Step 4: De-whitening(任意)

  このステップではすべての方向を元の分散にもどす、すなわちステップ 1 を適用した場合のみ意味がある。ある言語の単語ベクトルはその言語の元の分散だけでなく他の言語の元分散を利用することでも de-whitening を行うことができる。いずれにせよ、de-whitening を行う言語を A、利用する言語を B とすると以下の式が成り立つ。

$$W_{A(4)} = W_{B(2)}^T W_{B(1)}^{-} 1 W_{B(2)}$$

- Step 5: 次元削減 (任意)

  このステップでは単語ベクトルの成分を最初の n 個分だけ保つ。これは次の式で与えられる。

$$W_{X(5)=WZ(5)=(I_n 0)^T}$$

2.3 VecMap

VecMap は BWE を学習するためのフレームワークを実装したプログラムである [10]。BWE を構築するスクリプトや単語の翻訳、類似性/関連性、類推を評価するツールを内包している。

必要なもの

- Python3
- NumPy
- SciPy
- Cupy(CUDA をサポートしている場合)

使い方

BWE を構築するために、まず word2vec などを使ってそれぞれの言語で単一言語の単語ベクトルを訓練しておく。

マッピング

- Supervised
  大きい訓練辞書を持っている場合に推奨

```
>python3 map_embeddings.py
        --supervised
         TRAIN.DICT
         SRC.EMB TRG.EMB
         SRC_MAPPED.EMB TRG_MAPPED.EMB
```

- Semi-supervised
  seed dictionary を持っている場合に推奨

```
>python3 map_embeddings.py
        --semi_supervised
         TRAIN.DICT
         SRC.EMB TRG.EMB
         SRC_MAPPED.EMB TRG_MAPPED.EMB
```

- Identical
  seed dictionary を持っておらず、言語間で共通して使われている単語に依存する場合に推奨

```
>python3 map_embeddings.py
        --identical
         SRC.EMB TRG.EMB
         SRC_MAPPED.EMB TRG_MAPPED.EMB
```

- Unsupervised
  seed dictionary を持っておらず、言語間で共通して使われている単語に依存しない場合に推奨

```
>python3 map_embeddings.py
        --unsupervised
         SRC.EMB TRG.EMB
         SRC_MAPPED.EMB TRG_MAPPED.EMB
```

評価

以下のようにテスト辞書を使って BWE を評価することができる。

```
>python3 eval_translation.py
         SRC_MAPPED.EMB
         TRG_MAPPED.EMB
        -d TEST.DICT
```

上記のコマンドはスタンダードな最近傍法を使っている。よりよい結果を得るためには以下の CSLS 探索を代わりに使うことを推奨する。

```
>python3 eval_translation.py
         SRC_MAPPED.EMB
         TRG_MAPPED.EMB
        -d TEST.DICT
       --retrieval csls
```

CSLS は最近傍法と比べ実行速度が遅いため、–cuda オプションを使用するとよい。

テスト辞書に加え、単語の類似度でも BWE を評価することができる。

```
>python3 eval_similarity.py
        -l --backoff 0
         SRC_MAPPED.EMB
         TRG_MAPPED.EMB
        -i TEST_SIMILARITY.TXT
```

最後に、単一言語の単語類推を評価するツールも提供されている。

```
python3 eval_analogy.py
       -l SRC_MAPPED.EMB
       -i TEST_ANALOGIES.TXT
       -t 30000
```

2.4 BWEの応用

最後に、 BWE の応用について述べる。基本的に BWE は翻訳システムに利用できる [11]。翻訳以外の利用も可能である。例えば Chenggang Mi らはウイガル語の借用語を識別するために BWE を利用している [12]。具体的には借用語の候補リストを生成するために BWE を利用している。まず、ソース言語であるウイガル語のコーパスとターゲット言語であるドナー言語（中国語、アラビア語、ペルシャ語、ロシア語）の

コーパスを用い Closslingual Word Embeddings(以下 CWE) を学習する。そして、ウイガル語のコーパスから取り出した単語とドナー言語のコーパスから取り出した単語間の距離を構築した CWE から求め、その距離がスレッショルド ρ より小さいもの同士を取り出していくことでリストを生成する。

第3章　BERT

3.1　Attention Mechanism

3.1.1　seq2seq

seq2seq とはシーケンスのペアを大量に学習させることで、片方のシーケンスからもう一方を生成するモデルである。[13]
実用例としては以下のようなものがある。

- 翻訳：英語の文から、それをフランス語に翻訳した文を生成する。
- 構文解析：英語の文から、その文の構文木を生成する。
- 会話 bot：ある問い掛けから、それに対する返答を生成する。

このモデルの動きを英語から日本語への翻訳を例に考えると、「I have a pen」から、日本語でも英語でもない「意味のようなもの」を表す内部状態が得られ、その内部状態から「私はペンを持っている」が生成される。
seq2seq の問題点は上記の「意味のようなもの」を短い文でも長い文でも固定次元のベクトルの中に押し込まなくてはならず、文が長くなると翻訳の精度が落ちることである。

3.1.2　Attention Mechanism の原理

Attention Mechanism を数式で説明する。[14]

- 入力側
  - x_j：j 番目の入力単語
  - h_j：j 番目の入力に対応する隠れ層
- 出力側
  - y_t：t 番目の出力単語
  - s_t：t 番目の出力単語に対応する隠れ層
- Attention Mechanism

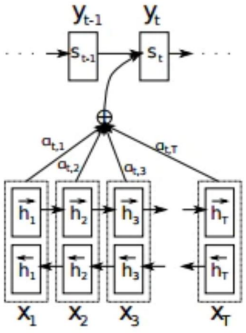

図 3.1: Attention モデルが単語 y_t を原文 $(x_1, x_2, ..., x_T)$ から生成する図

- α_{ij}：i 番目の単語に対して j 番目の単語が関連している確率もしくは結びつきの強さ

出力の隠れ層 s_i は次のように決定される。

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

c_i は次のように決定される。

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

α_{ij} はただ 1 つの j に対して大きい値（$\fallingdotseq 1.0$）を持ち、他はほぼ 0 になるよう学習される。つまり、c_i は i 番目の出力に関連度が高い入力の隠れ層になる。

ここで、α_{ij} は

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} = softmax_j(e_{ij})$$

$$e_{ij} = a(s_{i-1}, h_j)$$

であり、a はこの seq2seq もでる全体とともに学習されるフィードフォワードニューラルネットワークである。

まとめると、時刻 i の出力の隠れ状態 s_i を計算するとき、以下のように処理される。

FNN によって、s_{i-1} と各時刻 j の h_j を入力として、時刻 i の出力に最も関連のありそうな入力時刻 j が計算される（e_{ij}, α_{ij}）。そして、その時刻の入力の隠れ層が c_i として s_i への入力の一部となる。

3.2 Transformer

3.2.1 Attention

Attention は Query$Q$ と Key$K$ と Value$V$ の 3 つのベクトルを持っている [15]。より具体的には、Attention とは V 加重和であり、その加重は Q と K を使って計算される。

縮小付き内積 Attention

Q,K,V の正体は、Embedding X にそれぞれ重み W^Q,W^K,W^V を掛けあわせたものである。

縮小付き内積 Attention を文章で表すと、クエリと全キーの内積によりその関連度を計算し、$\sqrt{d_k}$（d_k は Q と K の次元）で割った後に、ソフトマックス関数を適用させることである。数式で表すと以下のようになる。

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}V$$

マルチヘッド Attention

各単語にたいして 1 組の Q,Key$K$,Value$V$ をもたせるのではなく、比較的小さい Q,Key$K$,Value$V$ をヘッドの数分つくり、それぞれのヘッドで潜在表現を計算する。最終的にそれらを 1 つのベクトルに落とすことによって獲得された潜在表現をその単語の潜在表現とする。

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$
$$where head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

3.2.2 Position-wise 順伝播ネットワーク

各ブロックの Attention 層のあとに入っている、各単語ごとに独立しているニューラルネットワーク。

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

3.3 BERT

3.3.1 概要

BERT の学習には以下の 2 段階がある [16]。

- 事前学習：ラベル無しデータを用いて、複数のタスクで事前学習を行う

- ファインチューニング：事前学習の重みを初期値として、ラベルありデータでファインチューニングを行う。

3.3.2 BERT の事前学習

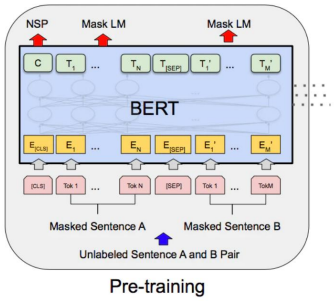

図 3.2: BERT の事前学習の様子

文を両方向で学ぶために BERT では 2 つの事前学習をおこなう。それが MLM と NSP である。

- Task1：Masked Language Modeling(MLM) 入力の 15 %のトークンを [Mask] トークンでマスクし、元のトークンを当て得るタスク。つまり、穴埋め問題である。この時、上図の T_i に Softmax を適用することでトークンを予測する。ただし、ファインチューニングと事前学習間の差異を緩和させるために、マスクするトークンに対して次の表のような処理を行う。

表 3.1: マスクするトークンに対しての処理

確率（%）	処理
80	[Mask] トークンで置き換える
10	ランダムに選んだトークンで置き換える
10	そのままにする

- Task2：Next Sentence Prediction(NSP) Q & A や自然言語推論などの文同士の関係を考慮する必要がある問題に対して対処するために、2 文選んでそれらが連続した文かどうかを当てるタスクを行う。

3.3.3 BERT のファインチューニング

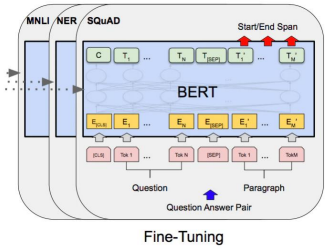

図 3.3: BERT のファインチューニングの様子

出力のうち C は識別タスク（EX. 感情分析）に使われ、T_i はトークンレベルのタスク（Ex.Q & A）に使われる。

第4章　提案手法

ここではまず、BERT を用いて英語のラベル付きの訓練文書をベクトル化し、それを利用して分類器を学習する。次に、日本語の文書を BERT でベクトル化したものをを日本語から英語への変換器で変換し、先の分類器で分類する。

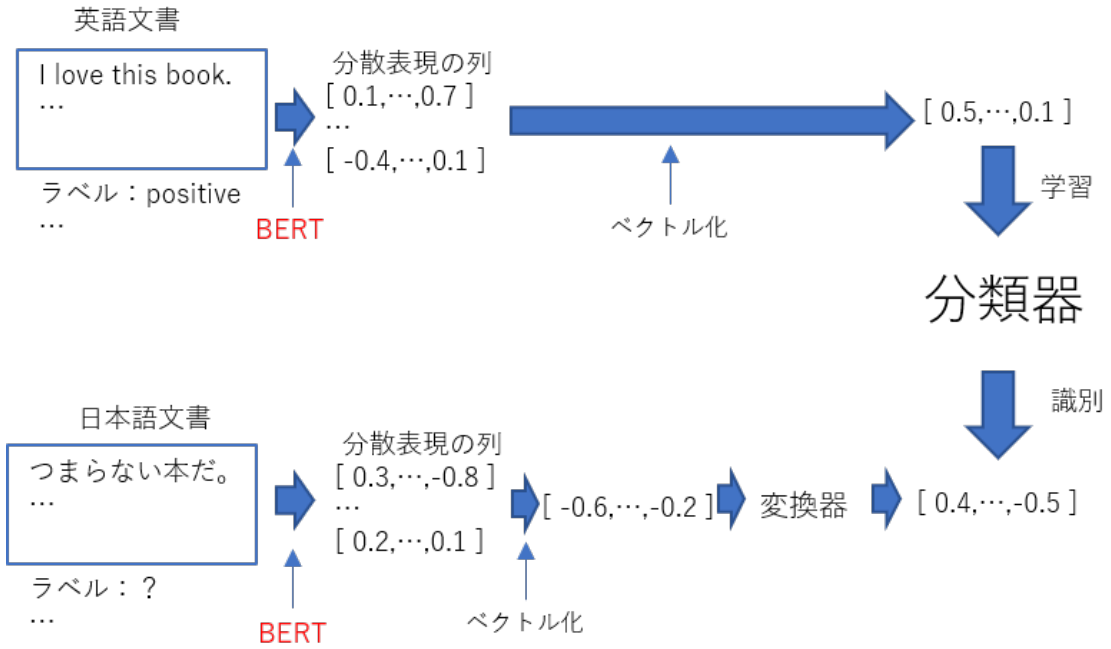

図 4.1: 提案手法の概要図

以下、変換器の構築とベクトル化の詳細を述べる。

4.1　変換器の学習

日本語から英語への変換器を以下のように学習する。対訳コーパスには田中コーパスを用いた。

学習には図 4.3 のネットワークを用いた。

ここで、$(E_1^j, ..., E_{762}^j), (E_1^e, ..., E_{762}^e)$ はそれぞれ対訳コーパスをベクトル化したものである。

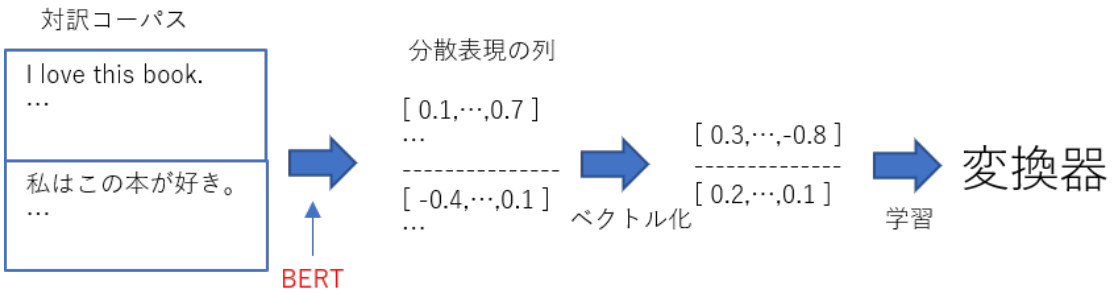

図 4.2: 変換器の学習

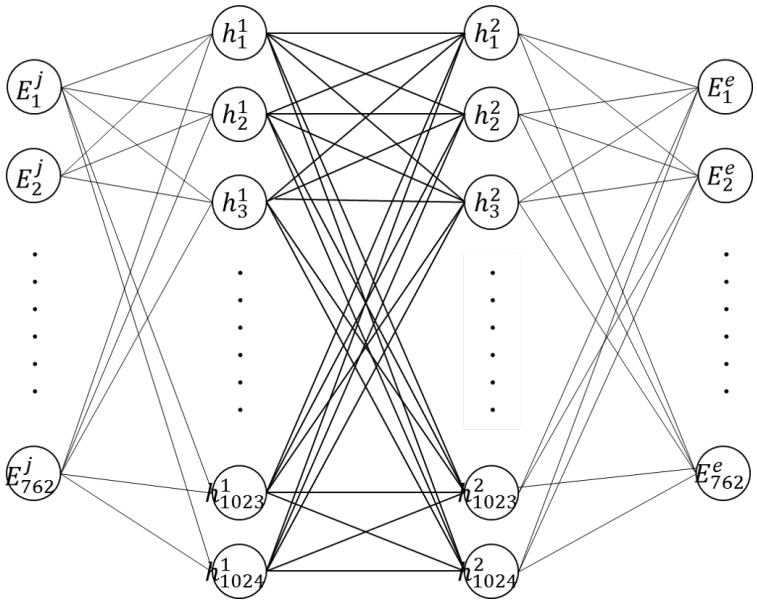

図 4.3: 学習に用いたネットワーク

4.2 文書のベクトル化

BERT を用いて文書中の各単語のベクトルを取得し以下のように文書全体のベクトルを求める。

$$S = \frac{w_1 + w_2 + ... + w_i}{i}$$

ここで、S は文のベクトル、w_i は i 番目の単語のベクトルを表している。

第5章　実験

5.1　データセット

感情分析のためのデータは以下のサイトのものを利用した。

`https://webis.de/data/webis-cls-10.html`

このデータは、日本語と英語でそれぞれ books、DVD、music の３つの領域を持ち、各データ（文書）数は 2000 である。また、日本語のテスト文書を英語に翻訳した文書を持つ。ラベルは１から５の５段階でつけられており、１と２を negative 、３から５を positive と判定した。

例

文書

要は短い英会話のヒアリング練習教材。 ステレオで聴くと、話し手が歩き回っているような臨場感があるのは面白いと思うが、星５つの評価は作為的な気がします。 何回も聴きこんでの感想ですが、脳の刺激がどうのこうの、といった薀蓄効果にも疑問を感じます。 ヒアリング力をつけたければ、やはり精聴（英会話を聴いて書取り）、多聴の地道なトレーニングは避けては通れないですし、それであれば別の教材の方がコストパフォーマンスはよいと思います。

ラベル

negative

5.2　翻訳を利用した感情分析

5.2.1　実験方法

まず、英語の訓練文書を Word2Vec を用いてベクトル化し、分類機を学習する。次に、日本語から英語に翻訳したテスト文書と英語のテスト文書を同様にベクトル化し、先の分類機で分類する。学習アルゴリズムには scikit-learn の SVM を用い、 C パラメータは 10 で固定した。

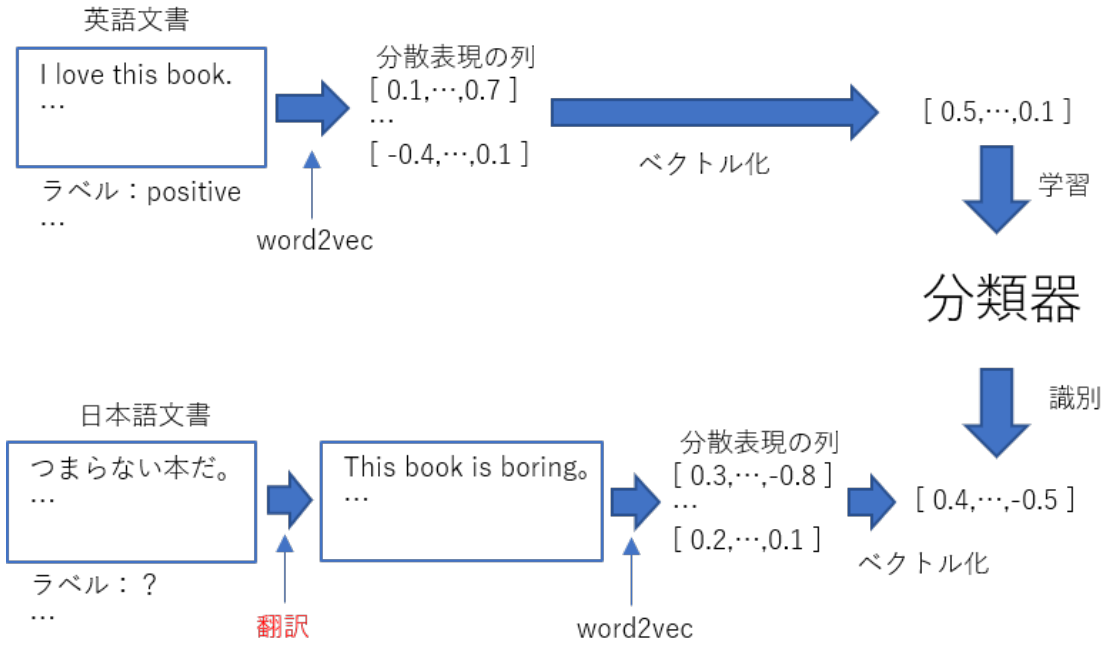

図 5.1: 翻訳を用いた感情分析の概要図

また、 Bag of Words(以下 BoW）を用いて文書をベクトル化した場合の実験も行った。学習アルゴリズムには scikit-learn の SVM を用い、 C パラメータは 100 で固定した。

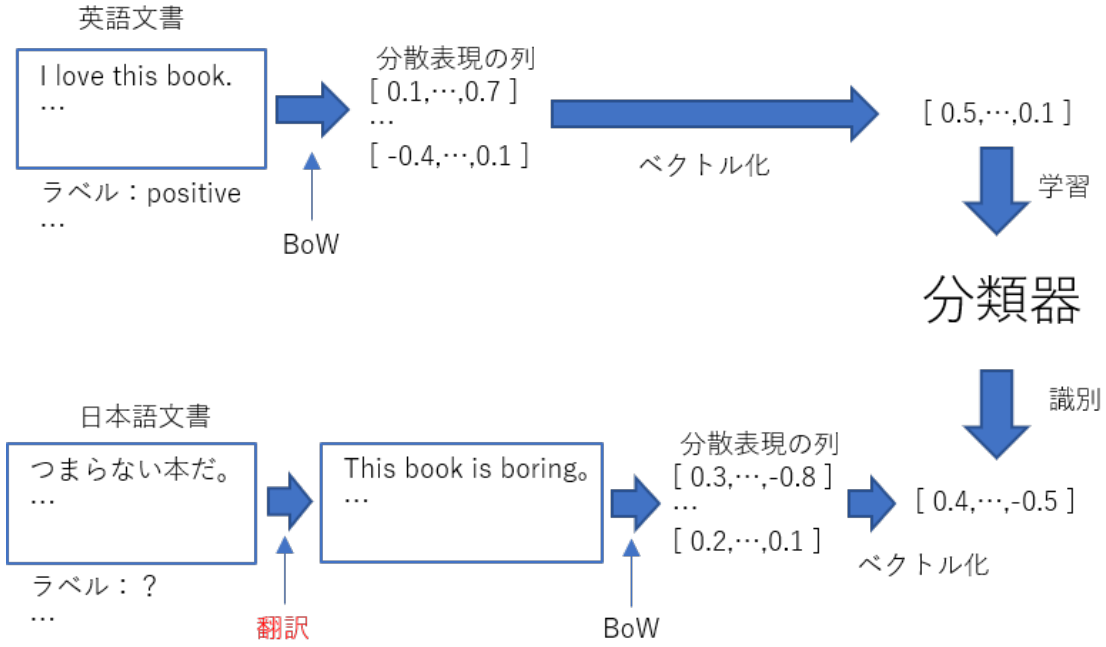

図 5.2: 翻訳を用いた感情分析の概要図（BoW）

5.2.2 実験結果

上記の実験の結果をそれぞれ表 5.1、5.2 に示す。

表 5.1: 翻訳による感情分析

テスト文書	books	DVD	music
日本語テスト文書を日英翻訳した文書	**0.69**	0.70	**0.72**
英語テスト文書	**0.77**	0.76	**0.78**

表 5.2: 翻訳による感情分析（BoW を用いた場合）

テスト文書	books	DVD	music
日本語テスト文書を日英翻訳した文書	0.66	**0.69**	0.70
英語テスト文書	**0.77**	**0.77**	0.74

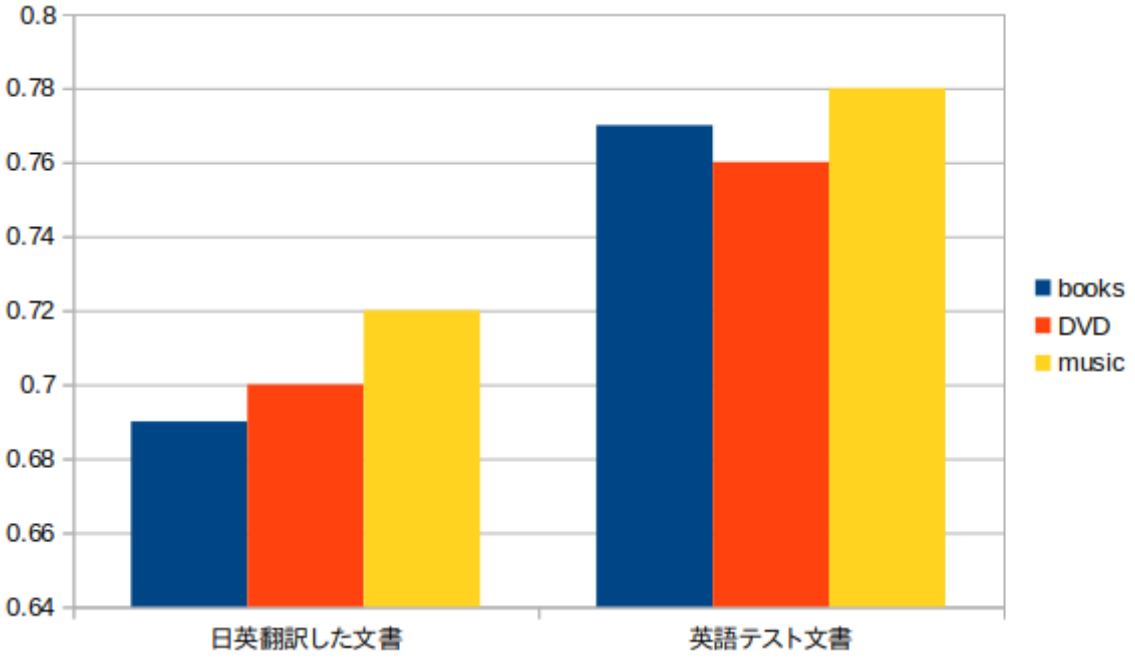

図 5.3: 翻訳による感情分析

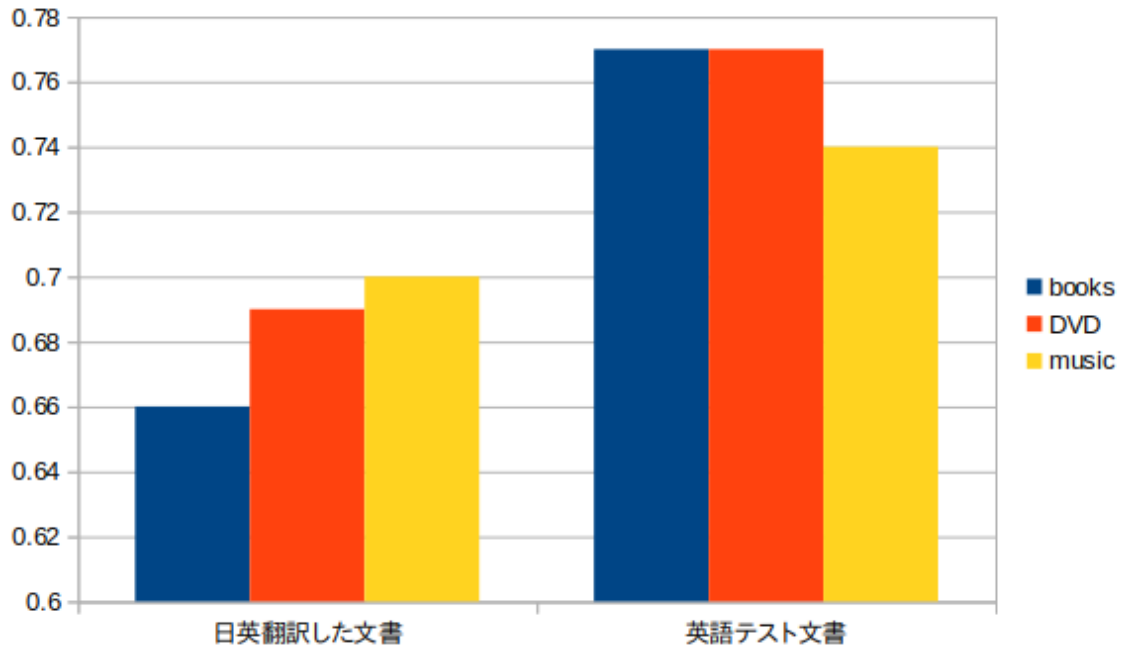

図 5.4: 翻訳による感情分析（BoW）

5.3 BWE と Word2Vec を利用した感情分析

5.3.1 実験方法

BWE を用いて文書をベクトル化し感情分析を行った。学習アルゴリズムには scikit-learn の SVM を用いた。 C パラメータは 1000 で固定した。

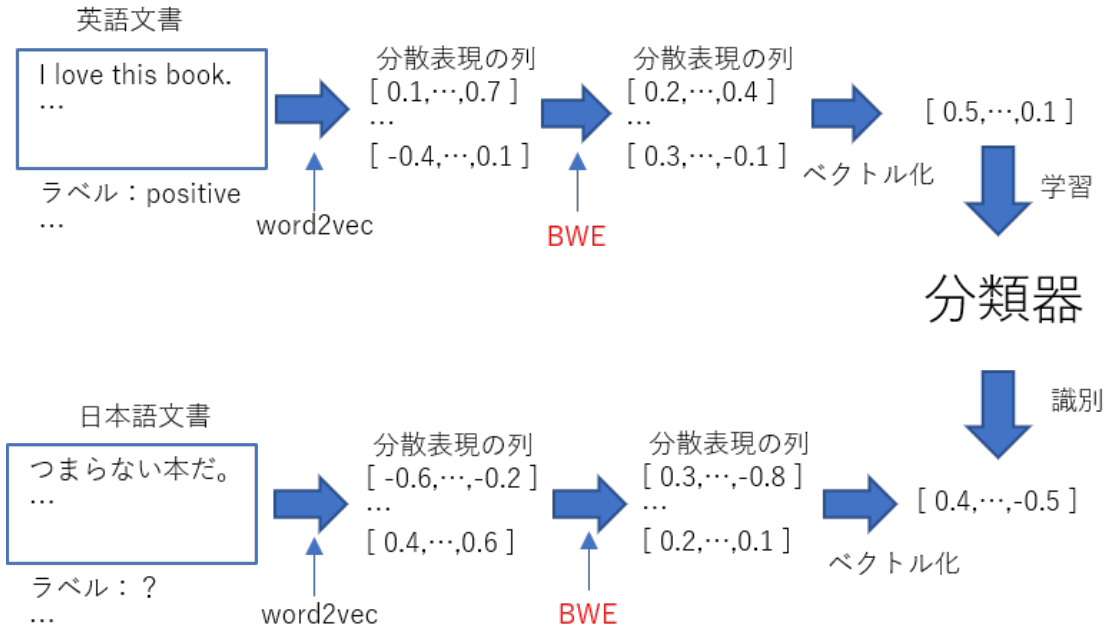

図 5.5: 翻訳を用いた感情分析の概要図

5.3.2 BWE の構築

VecMap [1]を利用して、英語と日本語の BWE を構築する。以下の手順により BWE を作成した。

1. 日本語の分散表現の作成

   gensim を用いて日本語の分散表現を作成した。コーパスは日本語の Wikipedia のデータを使用した[2]。

2. 英語の分散表現の作成

   gensim を用いて英語の分散表現を作成した。コーパスは英語の Wikipedia のデータを使用した[3]。

3. VecMap による BWE の作成

   VecMap を以下のように実行して、日本語と英語の BWE を作成した。

[1] https://github.com/artetxem/vecmap
[2] https://dumps.wikimedia.org/jawiki/latest/jawiki-latest-pages-articles.xml.bz2
[3] https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

```
> python3 map_embeddings.py
        --semi_supervised
          3000_common_words.en2ja
          wiki_en.emb
          wiki_ja.emb
          wiki_en_semi.bwe
          wiki_ja_semi.bwe
```

モードは Semi-supervised を利用した。また `3000_common_words.en2ja` は seed dictionary 、 `wiki_en.emb` は英語の分散表現、 `wiki_ja.emb` は日本語の分散表現、`wiki_en_semi.bwe` は英語の BWE 、`wiki_ja_semi.bwe` は日本語の BWE である。

5.3.3 実験結果

上記の実験の結果を表 5.3 に示す。

表 5.3: BWE と Word2Vec による感情分析

テスト文書	books	DVD	music
日本語テスト文書	0.64	0.69	0.70
英語テスト文書	0.75	0.74	0.76

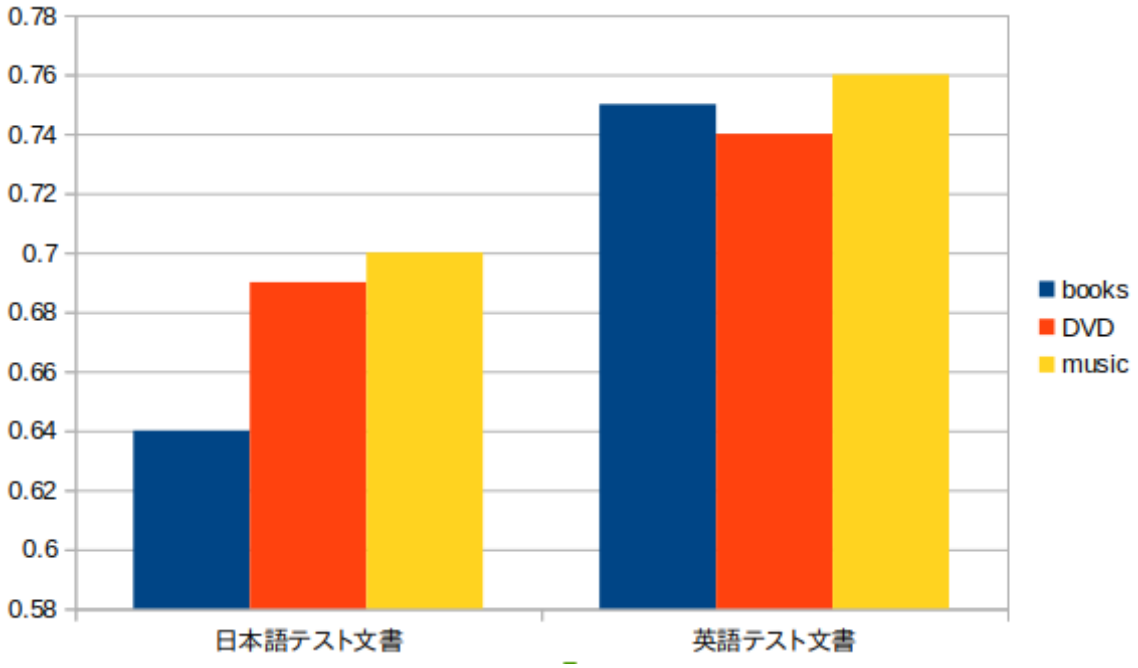

図 5.6: BWE と Word2Vec による感情分析

5.4　BWE と BERT を利用した感情分析（提案手法）

提案手法により感情分析を行った。分別器の学習アルゴリズムには 3 層のニューラルネットワークをを用いた。英語の文書を教師データとして次の 3 つのテスト文書に対して実験を行った。

- 日本語の文書（提案手法により変換）
- 英語の文書
- 日本語から英語に翻訳した文書

5.4.1　実験結果

結果を表??に示す。

表 5.4: BERT を用いた感情分析

テスト文書	books	DVD	music
日本語の文書	0.64	0.60	0.57
英語の文書	0.65	0.59	0.60
日英翻訳した文書	0.63	0.55	0.53

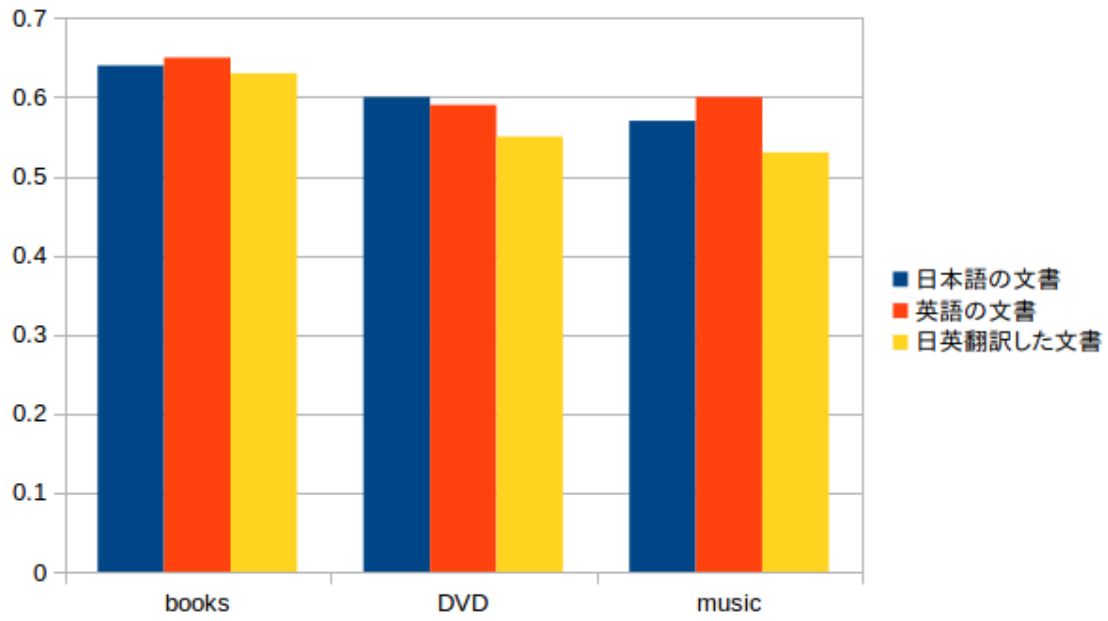

図 5.7: BERT を用いた感情分析

5.4.2 他手法との比較

日本語のテスト文書を感情分析した結果を手法ごとに比較した結果を表??に示す。

表 5.5: 手法ごとの比較

手法	books	DVD	music
翻訳 (分散表現)	0.69	0.70	0.72
翻訳 (BoW)	0.66	0.69	0.70
BWE+Word2vec	0.64	0.69	0.70
BWE+BERT	0.64	0.60	0.57

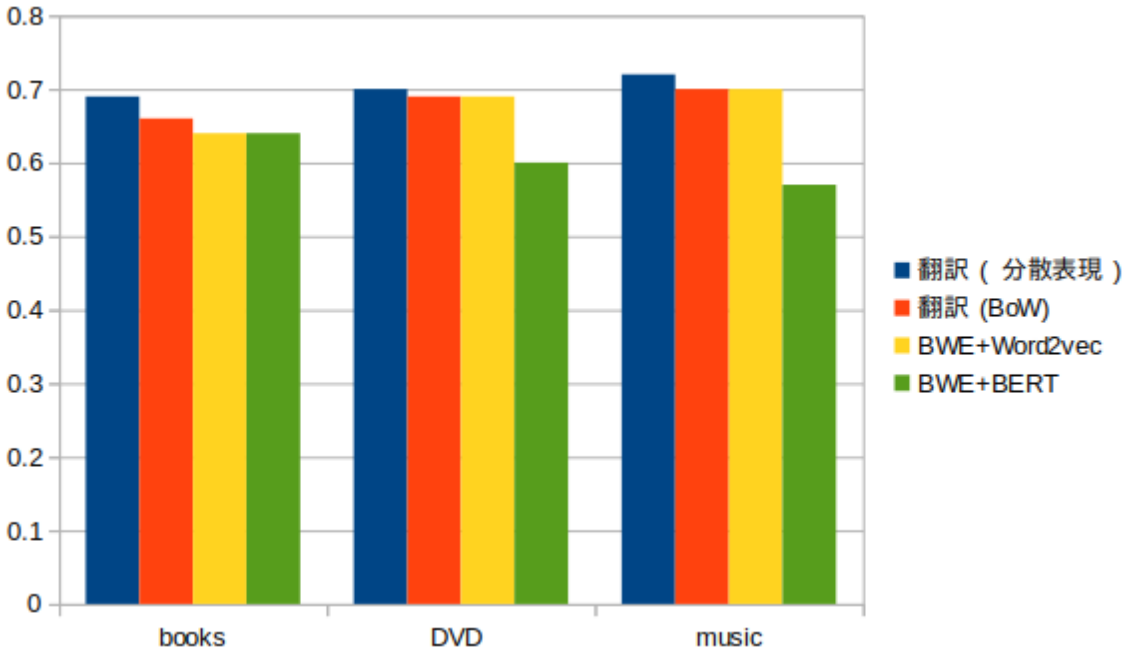

図 5.8: 手法ごとの比較

第6章　考察

まず、表5.1と表5.2を比較すると、結果があまり変わらないことがわかる。つまり、感情分析において、文のベクトル化の手法として単語の分散表現を用いる場合とBOWを用いる場合とでは精度の違いはほとんどないと言える。しかし、 BOW を用いる場合、ベクトルの特徴量は単語数に依存するため、単語数が多い場合は時間がかかってしまう。そのような場合は、分散表現を用いるのが良いだろう。

次に、表5.1から、「日英翻訳したテスト文書を使用した場合」と「英語のテスト文書を比較した場合」を比較すると、すべての領域において後者の方が明らかに正解率が高いことがわかる。つまり、英語の文書を用いて分類器を作成した場合、翻訳された英語の文書よりも、もともと英語で書かれた文書を分類する方が精度は高くなるということが言える。分類機の学習に用いた文書とベクトル化の方法は同じであるため、このような結果になる主な理由は、翻訳にあるのは確実である。おそらく、日本語にはあるが英語にはない表現が原因だろう。

例えば、「いただきます」や「ごちそうさま」を表す英語はない。もし、無理やり翻訳しようとすれば、”Let’s eat.(さあ食べましょう)” や ”I’m full.(満腹です)” となる。「いただきます」や「ごちそうさま」は相手を気遣い感謝を伝えるという意味合いが強いため、これらの訳は実際の意味合いとは異なっている。感情を表す表現としては、「もどかしい」などがある。これを翻訳しようとすると ”irritating(イライラする)” や ”be impatient(我慢できない)” など、「イライラする」といった意味合いになる。しかし、「もどかしい」は必ずしもそれに近い感情とは限らない。これらのような、英語で表現できない日本語の存在が精度を下げていると考えられる。

次に、表5.1と表5.3の結果を比べると、すべての領域において、 元々日本語のテスト文書に対する感情分析は、「翻訳を利用した場合」よりも「Word2Vec と BERT を利用した場合」の方が精度が少し低いことがわかる。前述した英語で表現できない日本語も原因の一つであると考えられるが、同じ意味の単語でも文脈によって意味合いにわずかな違いが生じることも原因だろう。

例えば、「カナダの首都はどこですか？」という文を英語に翻訳するとする。もし、「カナダ東部」という回答を期待するなら ”Where is the capital of Canada?” と翻訳されるが、「オタワ」という回答を期待する場合、 ”What is the capital of Canada?” と翻訳される。単語だけで見れば、通常「どこ」は ”where” と訳されるが、文の意味によっては ”what” と訳されることもある。このように、文の意味合いによって単語の訳し方も変わってくる。

また、表5.4より、BERT を用いた感情分析では、「テスト文書が英語の場合」に対し「テスト文書が日本語の場合」の精度があまり変わらない。しかし、表5.3より、Word2Vec を用いた感情分析では、「テスト文書が英語の場合」に対し「テスト文書が日本語の場合」の精度が6 10%低下することがわかる。つまり、BERT を用いた場合、

文書ベクトルを他言語へ変換することによる精度の低下が少ないといえる。その理由は、Word2Vec が単語そのものの意味をベクトル化するのに対し、BERT では文脈に沿った単語の意味をベクトル化するからであると推測できる。例えば、”mean”という単語は「意味する」、「卑劣な」、「平均」などの意味を持つが、Word2Vec で表すと全て同じ埋め込み表現になってしまう。さらにそれを変換するため、意味合いが異なる埋め込み表現なり、Word2Vec を用いた場合の精度が低下すると考えられる。

第7章　結論

本論文では教師データを利用しない感情分析を試みた。英語というメジャーな言語側での分類器の学習、つまりターゲット領域（日本語）の教師データ無しでの分類機の学習である。

実験では、BERT を用いた方法と Word2Vec を用いた方法を比較し、BERT を用いた場合のほうが、「テスト文書が英語の場合」に対する「テスト文書が日本語の場合」の精度の低下が少ないことが分かった。しかし、本論文の実験では、「テスト文書が英語の場合」の精度が低く、今後上げていくことが出来ればそれに伴って「テスト文書が日本語の場合」の精度も上がっていくと考えられる。今後はその方向で研究を進めていきたい。

参考文献

[1] 荘司響之介, 新納浩幸, 小宮嘉那子. Bilingual word embeddings によるターゲット言語の教師データを必要としない感情分析. 言語処理学会第 25 回年次大会, 2019.

[2] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

[3] Min Xiao and Yuhong Guo. Distributed word representation learning for cross-lingual dependency parsing. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pp. 119–129, 2014.

[4] Karl Moritz Hermann and Phil Blunsom. Multilingual distributed representations without word alignment. *arXiv preprint arXiv:1312.6173*, 2013.

[5] Alexandre Klementiev, Ivan Titov, and Binod Bhattarai. Inducing crosslingual distributed representations of words. *Proceedings of COLING 2012*, pp. 1459–1474, 2012.

[6] Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1006–1011. Association for Computational Linguistics, 2015.

[7] Angeliki Lazaridou, Georgiana Dinu, and Marco Baroni. Hubness and pollution: Delving into cross-space mapping for zero-shot learning. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 270–280. Association for Computational Linguistics, 2015.

[8] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 160–167, New York, NY, USA, 2008. ACM.

[9] Mikel Artetxe, Gorka Labaka, and Eneko Agirre. Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2289–2294, Austin, Texas, November 2016. Association for Computational Linguistics.

[10] Mikel Artetxe, Gorka Labaka, and Eneko Agirre. Generalizing and improving bilingual word embedding mappings with a multi-step framework of linear transformations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018.

[11] Will Y Zou, Richard Socher, Daniel Cer, and Christopher D Manning. Bilingual word embeddings for phrase-based machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1393–1398, 2013.

[12] Chenggang Mi, Yating Yang, Lei Wang, Xi Zhou, and Tonghai Jiang. Toward better loanword identification in uyghur using cross-lingual word embeddings. In *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 3027–3037, 2018.

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

付 録A　変換器を学習するプログラム

ソースコード A.1: henkan.py

```
1
2  import numpy as np
3  from keras import layers
4  from keras import models
5
6  def build_model():
7      model = models.Sequential()
8      model.add(layers.Dense(1024, activation='relu', input_shape=(768,)))
9      model.add(layers.Dense(1024, activation='relu'))
10     model.add(layers.Dense(768))
11     model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
12     return model
13
14 train_ja = np.load("tanaka-ja-5000.npy")
15 train_en = np.load("tanaka-en-5000.npy")
16
17 model = build_model()
18
19 model.fit(train_en, train_ja, epochs=100, batch_size=1, verbose=0)
20 model.save('henkanki.model')
```

付 録B　文をベクトル化するプログラム

B.1　日本語

ソースコード B.1: jsen2vec.py

```
1
2  import sys
3  from pyknp import Juman
4  import subprocess
5  import pandas as pd
6  import json
7  import numpy as np
8  import pickle
9
10 DIMENSION = 768
11
12 jumanpp = Juman()
13 argvs = sys.argv
14 argc = len(argvs)
15 errorfile = []
16 ngnumber = 0
17 matrix_ja = np.empty((0,DIMENSION))
18
19 for i in range(2):
20   si = str(i)
21   for j in range(10):
22     sj = str(j)
23     for k in range(10):
24       sk = str(k)
25       for l in range(10):
26         s = argvs[1]+si+sj+sk+str(l)+'.txt'
27         f = open(s,'r')
28         line = f.read()
29         f.close()
30         line = line.rstrip()
31         print(s+'\n')
32         try:
33           #日本語
34           result = jumanpp.analysis(line)
35           wakati_ja = result.mrph_list()[0].midasi
36           for n in range( 1, len(result.mrph_list())):
37             wakati_ja = wakati_ja + '␣' + result.mrph_list()[n].midasi
38
39           cmd = "echo␣"+'"'+wakati_ja+'"'+"␣>␣/home/syouji/bert-ja/
                 myinput.txt"
40           subprocess.call(cmd, shell=True)
41           print(wakati_ja)
42           cmd = "cat␣"+"/home/syouji/bert-ja/myinput.txt"
```

```
43            subprocess.call(cmd, shell=True)
44
45            cmd = "python3␣/home/syouji/bert-ja/extract_features.py␣--
                  input_file=/home/syouji/bert-ja/myinput.txt␣--output_file=/
                  home/syouji/bert-ja/myoutput.json␣--vocab_file=/home/syouji
                  /Japanese_L-12_H-\
46 768_A-12_E-30_BPE/vocab.txt␣--bert_config_file=/home/syouji/Japanese_L
       -12_H-768_A-12_E-30_BPE/bert_config.json␣--init_checkpoint=/home/
       syouji/Japanese_L-12_H-768_A-12_E-30_BPE/bert_model.ckpt␣--
       do_lower_\
47 case=False␣--layers=-1␣--max_seq_length=512␣--batch_size=8"
48
49            subprocess.call(cmd, shell=True)
50
51            ds = pd.read_json('/home/syouji/bert-ja/myoutput.json')
52
53            sum_vec = np.zeros(DIMENSION)
54            for n in range(1,len(ds['features'])-1): #cls と sep を除く
55              lst2arr = np.array(ds['features'][n]['layers'][0]['values'])
56              sum_vec += lst2arr
57            sen2vec = sum_vec / (len(ds['features'])-2)
58
59            #sen2vec = ds['features'][0]['layers'][0]['values']
60            sen2vec_nor_ja = sen2vec / np.linalg.norm(sen2vec) #正規化
61            matrix_ja = np.vstack([matrix_ja, sen2vec_nor_ja])
62
63          except Exception as e:
64            print(e,'erro␣occurred')
65            ngnumber = 1000*i + 100*j + 10*k + l
66            print(ngnumber)
67            errorfile.append(ngnumber)
68            ngnumber = 0
69
70 np.save(argvs[2],matrix_ja)
71 print(errorfile)
72 f = open(argvs[3],'wb')
73 pickle.dump(errorfile, f)
74 f.close()
```

B.2 英語

ソースコード B.2: esen2vec.py

```
1
2  import sys
3  from pyknp import Juman
4  import subprocess
5  import pandas as pd
6  import json
7  import numpy as np
8  import pickle
9
10 DIMENSION = 768
11
12 argvs = sys.argv
13 argc = len(argvs)
14 errorfile = []
15 ngnumber = 0
```

```
matrix_en = np.empty((0,DIMENSION))

for i in range(2):
  si = str(i)
  for j in range(10):
    sj = str(j)
    for k in range(10):
      sk = str(k)
      for l in range(10):
        s = argvs[1]+si+sj+sk+str(l)+'.txt'
        f = open(s,'r')
        line = f.read()
        f.close()
        line = line.rstrip()
        print(s+'\n')
        try:
          #英語
          wakati_en = line
          cmd = "echo␣"+'"'+wakati_en+'"'+"␣>␣/home/syouji/bert/myinput
              .txt"
          subprocess.call(cmd, shell=True)
          print(wakati_en)
          cmd = "cat␣"+"/home/syouji/bert/myinput.txt"
          subprocess.call(cmd, shell=True)

          cmd = "python3␣"+"/home/syouji/bert/extract_features.py␣"+"--
              input_file=/home/syouji/bert/myinput.txt␣"+"--output_file
              =/home/syouji/bert/myoutput.json␣"+"--vocab_file=/home/
              syouji/cased_L-12_H-\
768_A-12/vocab.txt␣"+"--bert_config_file=/home/syouji/cased_L-12_H-768
    _A-12/bert_config.json␣"+"--init_checkpoint=/home/syouji/cased_L-12
    _H-768_A-12/bert_model.ckpt␣"+"--do_lower_case=False␣"+"--layers
    =-\
1␣"+"--max_seq_length=512␣"+"--batch_size=8"
          subprocess.call(cmd, shell=True)

          ds = pd.read_json('/home/syouji/bert/myoutput.json')
          sum_vec = np.zeros(DIMENSION)
          for n in range(1,len(ds['features'])−1): #cls と sep を除く
              lst2arr = np.array(ds['features'][n]['layers'][0]['values'])
              sum_vec += lst2arr
          sen2vec = sum_vec / (len(ds['features'])−2)
          sen2vec_nor_en = sen2vec / np.linalg.norm(sen2vec) #正規化
          matrix_en = np.vstack([matrix_en, sen2vec_nor_en])

        except Exception as e:
          print(e,'erro␣occurred')
          ngnumber = 1000∗i + 100∗j + 10∗k + l
          errorfile.append(ngnumber)
          ngnumber = 0

np.save(argvs[2],matrix_en)
print(errorfile)
f = open(argvs[3],'wb')
pickle.dump(errorfile, f)
f.close()
```