平成30年度 茨城大学工学部情報工学科
卒業研究論文

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

平成31年2月5日

情報工学科
15T4034S 荘司 響之介
新納 浩幸 教授

平成 30 年度茨城大学工学部情報工学科卒業研究論文

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

15t4034S 荘司 響之介
新納 浩幸 教授

論文要旨

本論文では 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 を利用する。

本論文では英語のラベル付き文書を BWE 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 BWE を用いてベクトル化し、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 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）と比較するために、日本語のテスト文書を英語に自動翻訳することで英語側で作った分類器を利用する手法を試した。その結果 BWE を利用した英日の感情分析は、翻訳を利用した場合よりも精度はわずかに低かった。その原因は、英語で表現できない日本語が存在することや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だと考えられる。

提案手法は翻訳を用いた場合と比べ、翻訳する手間がかからないという点では優位であるといえるが精度はわずかに劣る。精度を改善するためには文をベクトルする際、文脈による単語の意味合いの違いを表現する必要があるため、今後はその方向で研究を進めたい。

目 次

表 目 次

図 目 次

第1章　はじめに

1.1　概要

本論文では 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 を利用する [12]。

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。

本論文では英語のラベル付き文書を BWE 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 BWE を用いてベクトル化し、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 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）と比較するために、日本語のテスト文書を英語に自動翻訳することで英語側で作った分類器を利用する手法を試した。

1.2　構成

本論文では、教師データを利用せずに感情分析を行うために BWE を用いる。２章で BWE についての説明を行い、３章ではそれを用いた感情分析について具体的な手法を説明する。

第2章　BWE

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 [8]。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 [9]。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 [8] 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 [10] 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。

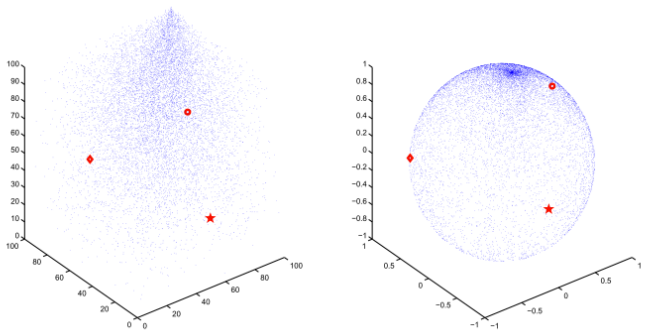

図 2.1: 正規化する前 (左) と後 (右) の単語表現

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 [6] 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 [3]) 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。

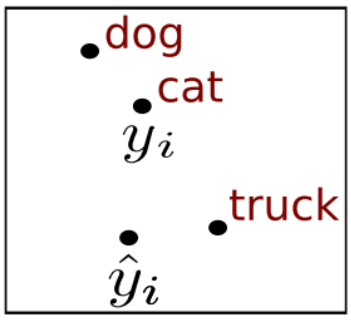

図 2.2: "cat" の負例として"dog"ではなく intruder である"truck"が選ばれる

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 [1] 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 [2] 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 [2]。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 [7]。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　提案手法

ここではまず英語と日本語の BWE を構築する。次に構築した BWE を用いて、英語の教師データ（ラベル付き文書）を BWE の列で表現する。次にこの BWE の列をベクトル化する。これによって分類器を学習する。識別では日本語のテストデータ（ラベルなし文書）を BWE の列で表現し、英語の場合と同様に、それをベクトル化する。その文書ベクトルを先に構築した分類器を用いて識別する (図 3.1 参照)。

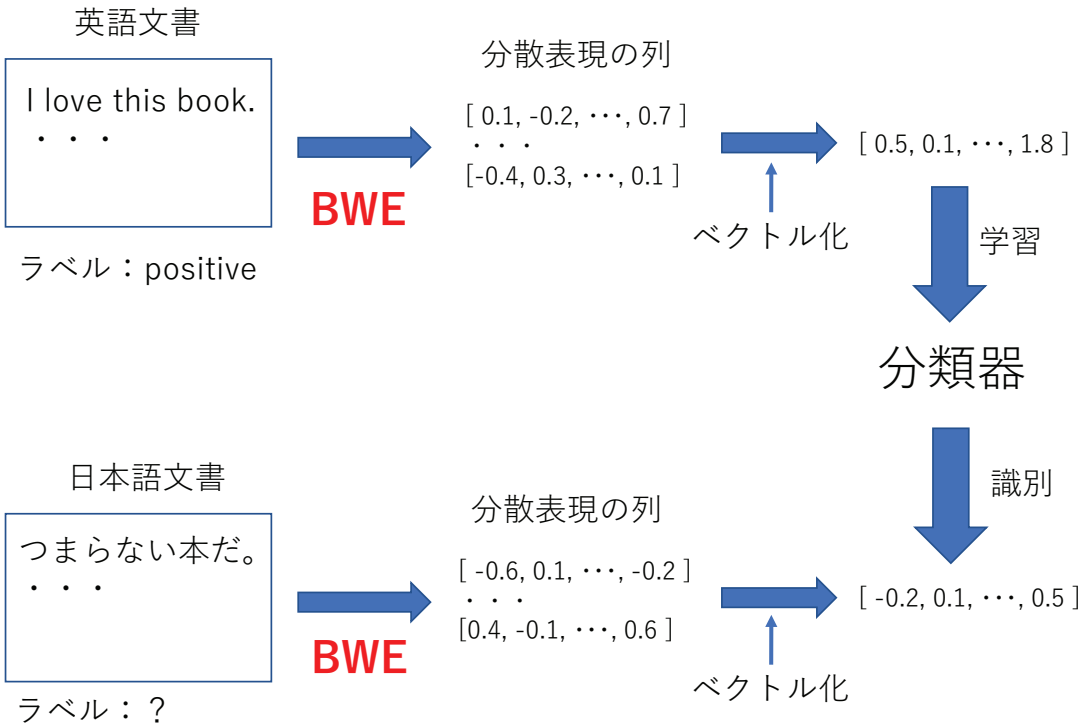

図 3.1: 提案手法の概要図

以下、BWE の構築とベクトル化の詳細を述べる。

3.1　BWE の構築

本論文では 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] https://github.com/artetxem/vecmap

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。

```
> 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。

3.2 BWE による文書のベクトル化

作成した BWE を用いて以下の手法で文書をベクトル化する。

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

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。

[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

第4章　実験

4.1　英日の感情分析データ

BWE を利用した感情分析の実験のために、以下のサイトから英語と日本語の感情分析のデータをダウンロードした。

`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。

4.2　翻訳を利用した感情分析

BWE による文書のベクトル化と同様に、英語の分散表現を用いて英語の訓練文書と日本語から英語に翻訳したテスト文書をベクトル化した。比較のため、英語の訓練文書を用いて学習した分類器を利用して、英語のテスト文書の感情分析を行った。学習アルゴリズムには scikit-learn の SVM を用い、C パラメータは 10 で固定した。結果を表 4.1 に示す。

表 4.1: 翻訳による感情分析（英語の分散表現）

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**

さらに、　Bag of Words (以下 BoW) を用いて翻訳を利用した英日の感情分析も行った。こちらも比較のため英英の感情分析も行った。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。結果を表 4.2 に示す。

表 4.2: 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

4.3 BWE を利用した感情分析

BWE を用いて文書をベクトル化し、英日の感情分析を行った。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。結果を表 4.3 に示す。

表 4.3: BWE 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

また、他の手法と比較した結果を表 4.4 と図 4.1 に示す。

表 4.4: 英日の感情分析の手法ごとの比較

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	0.64	0.69	0.70

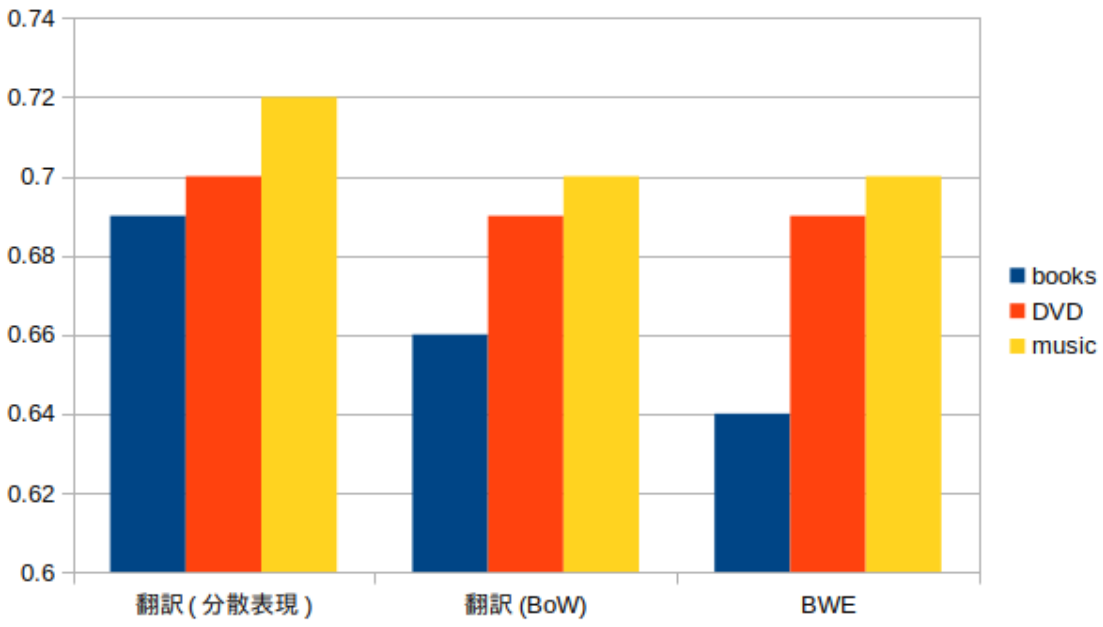

図 4.1: 英日の感情分析の手法ごとの比較

第5章　考察

まず、表 4.1 と表 4.2 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 4.1 から、翻訳を利用した英日の感情分析と英語の分散表現を利用した英英の感情分析を比較すると、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 4.1 と表 4.3 の結果を比べると、すべての領域において、 BWE を利用した英日の感情分析は、翻訳を利用した場合よりも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。

しかし、今回は文をベクトル化する際、単純に単語ベクトルの平均をとる方法を用いたため、文ごとの単語の意味合いの違いを表現することができなかったと考えられる。

第6章　おわりに

本論文では 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た。BWE を利用した英語などの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、翻訳を用いた感情分析と比較することで、提案手法の優位性を示した。しかし、翻訳する手間がかからないという意味では提案手法が優位であるが、精度はわずかに劣る結果となった。これは、文をベクトルする際、文脈による単語の意味合いの違いを表現できなかったためだと考えられる。ある単語の前後の単語にも注目することでその問題を解決できる可能性があるため、今後はその方向で研究を進めたい。

参考文献

[1] 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.

[2] 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.

[3] 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.

[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] 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.

[7] 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.

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

[9] 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.

[10] 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.

[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] 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.

付 録A　辞書を作成するプログラムのソースコード

ソースコード A.1: 辞書作成

```
1  from splinter import Browser
2  import pandas as pd
3  import sys
4
5  argvs = sys.argv
6  argc = len(argvs)
7
8  fr = open(argvs[1],'r')
9  fw = open(argvs[2],'w')
10
11 browser=Browser('firefox')
12
13 line = fr.readline()
14 while line:
15   line=line.rstrip()
16
17   browser.visit('http://www.bing.com/translator/')
18
19
20   search_bar_xpath='//*[@id="t_sv"]'
21   search_bar=browser.find_by_xpath(search_bar_xpath)
22   search_bar.fill(line)
23
24   try:
25     search_results_word_xpath='/html/body/div[4]/div[3]/div[5]/div[2]/
           table/tbody/*/td[1]'
26     search_results_word=browser.find_by_xpath(search_results_word_xpath)
27
28     if len(search_results_word)<=4:
29       l=len(search_results_word)
30     else:
31       l=4
32
33     for i in range(l):
34       fw.write(line+'␣'+search_results_word[i].text+'\n')
35   except Exception:
36     pass
37
38   line = fr.readline()
39 fr.close
40 fw.close
```

付 録B　文をベクトル化するプログラムのソースコード

B.1　日本語

ソースコード B.1: 日本語の文をベクトル化

```
1  import MeCab
2  import gensim
3  import numpy as np
4  from gensim.models import word2vec
5  import sys
6  import warnings
7
8  warnings.filterwarnings("error")
9  mt = MeCab.Tagger('')
10 mt.parse('')
11 model = gensim.models.KeyedVectors.load_word2vec_format("./en_ja_bwe/
       wiki_ja_semi_imp.bwe",binary=False)
12 #model = word2vec.Word2Vec.load("./wiki_ja/wiki_ja.model")
13
14 # テキストのベクトルを計算
15 def get_vector(text):
16     sum_vec = np.zeros(200)
17     word_count = 0
18     node = mt.parseToNode(text)
19     while node:
20         fields = node.feature.split(",")
21         # 名詞、動詞、形容詞に限定
22         if fields[0] == '名詞' or fields[0] == '動詞' or fields[0] == '形容詞':
23             try:
24                 sum_vec += model[node.surface]
25                 word_count += 1
26             except Exception:
27                 pass
28         node = node.next
29     try:
30       return sum_vec / word_count
31     except Exception as e:
32       print(e,'error␣occurred')
33       return np.zeros(200)
34
35 # cos 類似度を計算
36 def cos_sim(v1, v2):
37     return np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))
38
39
40 if __name__ == "__main__":
41   argvs = sys.argv
42   argc = len(argvs)
```

```
43
44    matrix = np.empty((0,200))
45
46    for i in range(2):
47      si = str(i)
48      for j in range(10):
49        sj = str(j)
50        for k in range(10):
51          sk = str(k)
52          for l in range(10):
53            s = argvs[1]+si+sj+sk+str(l)+'.txt'
54            f = open(s,'r')
55            text = f.read()
56            f.close()
57            vec = get_vector(text)
58            if vec.all() == 0:
59              print("error:"+s)
60            else:
61              matrix = np.vstack([matrix,vec])
62
63    np.save(argvs[2],matrix)
```

B.2 英語

ソースコード B.2: 英語の文をベクトル化

```
1  #1文が入っているディレクトリ　ex)cls−amazon/en/books/train/
2  #2行列の保存先
3  #3ベクトル化できなかった文の番号のリストの保存先
4
5  from polyglot.text import Text
6  import numpy as np
7  import warnings
8  import gensim
9  from gensim.models import word2vec
10 import sys
11 import pickle
12
13 warnings.filterwarnings("error")
14
15 #bwe
16 #model = gensim.models.KeyedVectors.load_word2vec_format("./en_ja_bwe/
       wiki_en_semi_imp.bwe",binary=False)
17
18 #英語の分散表現
19 model = gensim.models.KeyedVectors.load_word2vec_format("./wiki_en/wiki_en.
       emb",binary=False)
20
21 #model = word2vec.Word2Vec.load("./wiki_en/wiki_en.model")
22
23 def get_vector(text):
24    sum_vec = np.zeros(200)
25    word_count = 0
26    node = Text(text)
27    for node in node.pos_tags:
28      if node[1][0] == 'N' or node[1][0] == 'V' or node[1][0] == 'A':
29        try:
```

```
30            sum_vec += model[node[0]]
31            word_count += 1
32          except Exception:
33            pass
34      try:
35        return sum_vec / word_count
36      except Exception as e:
37        print(e,'error occurred')
38        return np.zeros(200)
39
40  if __name__ == "__main__":
41      argvs = sys.argv
42      argc = len(argvs)
43
44      matrix = np.empty((0,200))
45
46      ngnumber=0
47      ngnumbers=[]
48
49      for i in range(2):
50        si = str(i)
51        for j in range(10):
52          sj = str(j)
53          for k in range(10):
54            sk = str(k)
55            for l in range(10):
56              s = argvs[1]+si+sj+sk+str(l)+'.txt'
57              f = open(s,'r')
58              text = f.read()
59              f.close()
60              try:
61                vec = get_vector(text)
62                if vec.all() == 0:
63                  print("error:"+s)
64                  ngnumber = 1000*i + 100*j + 10*k + l
65                  ngnumbers.append(ngnumber)
66                  ngnumber = 0
67                else:
68                  matrix = np.vstack([matrix,vec])
69              except Exception as e:
70                print(e,'error occurred'+s)
71                ngnumber = 1000*i + 100*j + 10*k + l
72                ngnumbers.append(ngnumber)
73                ngnumber = 0
74            print(si+sj+sk+'\n')
75
76      f = open(argvs[3],'wb')
77      pickle.dump(ngnumbers, f)
78      f.close()
79      print(ngnumbers)
80      print(len(ngnumbers))
81      np.save(argvs[2],matrix)
82      print(matrix.shape)
```