令和 4 年度茨城大学工学部情報工学科

卒業研究論文

ゼロショット文書分類におけるラベル名の性質の検討

所属　　情報工学科

著者　及川翔矢 (18T4017L)

指導教員　　新納浩幸教授

令和 5 年 2 月 3 日 (金)

令和 4 年度茨城大学工学部情報工学科　卒業研究論文

ゼロショット文書分類におけるラベル名の性質の検討

著者

及川翔矢 (18T4017L)

指導教員

新納浩幸教授

論文要旨

本論文ではゼロショット学習による文書分類の精度を向上させるラベル名の性質の検討、及びラベル名の自動付与を行う。

ゼロショット学習とは事前に知られた知識を利用することで、学習データに存在しないパターンを予測することができる機械学習手法である。そのため、大量のラベル付きデータを必要とする従来の機械学習手法に変わる手法として注目を集めている。しかしながらゼロショット学習による文書分類は、利用する情報の少なさのために問題を持つ。

先行研究におけるゼロショット学習による文書分類は、文書とラベル名をベクトル化し、二つのベクトル同士の近さを計測することで分類を行う。しかし近さのみを分類の基準とすると、適切な分類ができるラベル名を与えていない時に、文章を誤ったラベルに割り当ててしまい分類精度を低下させる原因となる。そこで本論文ではラベル名を独自のラベル名に変更することでゼロショット学習による文書分類の精度が向上することを確認し、同時に分類精度が向上するラベル名の性質を考察を目的とした研究を行う。

実験では、独自付与のラベル名には、各クラスに所属する文書の意味を説明する文章を与えた。自動付与によるラベル名には、文の特徴を捉える手法である Bag-of-Words と TF-IDF、加えて中心の文章を利用し文のラベル名を与えた。分類の結果、独自付与のラベル名と自動付与のラベル名のいずれでも精度の向上が見られ、ラベル名変更の有効性を確認できた。また、実験の結果から、文書分類の精度が向上するラベル名は、カテゴリの意味を説明する上位語下位語を考慮した一般名詞であると考察した。

目次

表目次

図目次

第 1 章

序論

1.1 研究の概要

文書分類とは、自然言語で与えられた文書に対して、文書を特定の分類体系に自動的に分類する処理を指す。人間の代わりに計算機に文書を分類させることは効率面などで重要である。しかし未知のカテゴリに属する文書を新たに機械に分類させる場合、大量のラベル付き学習データを用いて再学習を行わせる必要がある。これは、新しいパターンの大規模なデータが用意できない問題や再学習を行う際の時間的なコストがかかる問題を生じさせる。近年、このような問題を解決する手法の一つとしてゼロショット学習が注目されている。

ゼロショット学習とは事前に知られた知識を利用することで、学習データに存在しないパターンの予測をできるようにする機械学習手法である。先行研究 [1] におけるゼロショット学習による文書分類では、文書とラベル名をベクトル化し、二つのベクトル同士の近さを計測して、最も近いラベル名を分類先のラベルとすることで文書分類を行う実験がされている。

しかし、近さのみを分類の基準とすると、適切な分類ができるラベル名を与えていない時に、文章を誤ったラベルに割り当ててしまい分類精度を低下させる原因となる。一例を挙げると、本実験で用いるデータセットである livedoor ニュースコーパスには「エスマックス」というカテゴリがある。このカテゴリは主にスマートフォンやパソコンといった情報端末についてのカテゴリである。ところが、「エスマックス」と情報端末には言葉や意味としての関係性がないため、エスマックスと情報端末の 2 つが結びつく前提知識は存在しない。そのため、ベクトル化をして近さを計測してもゼロショット学習で

は正しく分類を行うことができない。

また、ゼロショット学習による文書分類の精度の向上には、文書やラベル名に意味を捉えたベクトル化を行う必要がある。ゼロショット学習による文書分類では、ベクトル化された文とラベル名の近さを測り分類を行う。したがって、意味が近い文とラベル名のベクトル同士は似ているベクトルに、意味が遠い文とラベル名のベクトル同士は違ったベクトルにする必要がある。BERT は、単語間の文脈をとらえたベクトル化を行うことができ、意味の近さを考慮したベクトル化を行うことが可能である。

本論文では、ゼロショット学習による文書分類において、文書とラベル名の近さを測り分類する際に分類精度が向上するラベル名の性質を考察を目的とした研究を行う。ラベル名を独自に与えたラベル名に変更することでゼロショット学習による文書分類の精度が向上することを確認し、ラベル名の自動付与の研究を行うことで、これまで人手によりつけられていたラベル名を機械的に行うことができるようになる。

実験では、ラベル名に独自付与のラベルと自動付与のラベルを与え文書分類を行った。独自付与のラベル名には、各クラスに所属する文書の意味を説明する文を与え文書分類を行った。自動付与によるラベル名には、Bag-of-Words、TF-IDF、中心の文章を利用して文のラベル名を与え文書分類を行った。また、文書のベクトル化の際には、BERT をファインチューニングし文の意味を捉えた分散表現を獲得できる Sentence-BERT を利用した。

1.2　本論文の構成

第 2 章では、関連研究について述べる。第 3 章では、独自付与のラベル名と自動付与のラベル名の付け方に関する提案手法を説明する。第 4 章では、与えたラベル名でゼロショット学習による文書分類を行った結果について述べる。第 5 章では、結果の分析とゼロショットにおける精度が向上するラベル名の性質を考察する。第 6 章では、本論文の結論を述べる。

第 2 章

関連研究

2.1 Bag-of-Words

文書分類において、文書をコンピュータに分類させるために、コンピュータが計算可能な形式であるベクトル化の必要がある。文書をベクトル化する手法として、Bag-of-Words がある。Bag-of-Words は、ベクトルの各次元に単語を割り当て、文書中の単語の出現回数をベクトルの値とすることで、文書をベクトル化するものである。Bag-of-Words による、文のベクトル化の手順を以下に示す。

1. 単語にインデックスを割り当てる
2. 各文ごとに単語の出現回数を数え、出現した単語に割り当てられたインデックスに 1 を足す
3. 手順 1 と 2 より作成した、各文の各単語の登場回数のベクトルをその文のベクトルとする

Bag of Words によるベクトル化の例を以下に示す。次の二つの文が与えられた時、これらの文に出現した単語を基に、表 2.1 のような対応表が作られ、表 2.2 のようなベクトルが生成される。

文 1： 単語の出現回数を数える

文 2： 出現回数のベクトルを作る

表 2.1: 単語に割り当てるインデックス

	単語	出現	回数	ベクトル	数える	作る	の	を
ID	0	1	2	3	4	5	6	7

表 2.2: Bag-of-Words によるベクトルの例

	単語	出現	回数	ベクトル	数える	作る	の	を
ID	0	1	2	3	4	5	6	7
文 1	1	1	1	0	1	0	1	1
文 2	0	1	1	1	0	1	1	1

2.2　TF-IDF

TF-IDF は、Bag-of-Words と同様、文書のベクトル化に用いられる。Bag-of-Words との違いは、ベクトルの各次元の値を重要な単語の寄与を大きく、重要でない単語の寄与を小さくすることで生成することである。具体的には、以下 2 つのアイディアに基づいてベクトルを生成する。

1. ある文中で、他の単語と比較して出現回数が多い単語は、その文の意味を表現するために重要な単語である。
2. 多くの文に広く出現する単語は一般的な単語であると考えられ、個々の文の意味を表現する上で重要な単語ではない

TF-IDF は、単語の出現回数を表す TF と、その単語を含む文の偏りを表す IDF の 2 つに基づいて計算される。TF-IDF による文のベクトル化の手順を以下に示す。

1. **TF を計算する**

   TF は単語の種類ごとに決まる値で、各文における各単語の出現回数を表す。文 d における各単語 t の TF は以下のように求められる。

   $TF(t, d) =$ 文 d に登場する単語 t の数

2. **DF を計算する**

DF は単語の種類ごとに決まる値で、単語ごとにその単語を含む文の数を求め、文の総数で割る事で得られる。単語 t の DF は以下のように求められる

$DF(t) = \frac{t\text{ を含む文の数}}{\text{文の総数}}$

3. **IDF を計算する**

IDF とは DF の逆数の対数である。

$IDF(t) = \log \frac{1}{DF(t)} = \log \frac{\text{文の総数}}{t\text{ を含む文の数}}$

4. **TF と IDF を掛け合わせる**

手順 1 で求めた TF と手順 3 で求めた IDF を乗じた値が TF-IDF である

$TFIDF(t,d) = TF(t,d) \cdot IDF(t)$

2.3　単語分散表現

分散表現とは、1 つの単語の意味を低次元の実数値ベクトルで表現することである。Bag-of-Words や TF-IDF は単語の出現回数を数えることでベクトルを作成する。しかし、単語の出現回数だけでは単語間の関係や意味をとらえることができず、結果として文の意味をとらえることはできない。Mikolov が提案した単語の分散表現のモデルである Word2Vec [3] は、大規模なコーパスで用いて単語を学習することで、単語間の意味をとらえた文のベクトルを生成することができる。

単語の分散表現は、単語の意味的な類似性を表すだけでなく、以下のような性質を持つことが知られている。

$$\text{日本} - \text{東京} = \text{イギリス} - \text{ロンドン} \tag{2.1}$$

この例は、「日本-東京」と「イギリス-ロンドン」の単語対に現れる意味的な関係性 (国-首都) が、単語の分散表現の差として現れることが分かる。このように単語の分散表現は、分散表現の加減算と意味的な加減算が一致するという性質である加法構成性が成り立つ。

2.4　BERT

BERT(Bidirectional Encoder Representations from Transformers) [4] は、2018 年に Google より発表されたされた汎用型自然言語処理モデルであり、様々な自然言語処理

のタスクで最高精度を達している。BERT の特徴は、双方向の Transformers [5] で学習することで、文脈を深く考慮した分散表現を生成できる点が挙げられる。文脈を考慮した分散表現とは、同じ単語でも周りの文章が変わるとそれに応じたベクトルを出力することである。BERT は、Attention という手法により離れた位置にある情報も適切に取り入れることができるため、文脈を深く考慮した分散表現を生成できる。

BERT の学習は、事前学習とファインチューニングという 2 つの過程からなる。事前学習では、大量の文書データを用いて汎用的な言語パターンを学習する。ファインチューニングでは、比較的少数のラベル対データを用いて、特定のタスクに特化して学習する。

2.4.1　入出力

BERT の入力には、以下で示すように単一の文章や文章のペアを単語列に変換して行う。単一の文章を入力する時には、文章をトークン化し、列の先頭に特殊トークンである [CLS] を、列の末尾に特殊トークンである [SEP] を加える。例として「今日の天気は晴れです。」をトークン列にすると以下のようになる。

’[CLS]’, ’ 今日’, ’ の’, ’ 天気’, ’ は’, ’ 晴れ’, ’ です’, ’。’, ’[SEP]’

また、文章のペアを入力する時には 2 つの文章のトークン列を並列に結合する。その際、文章の間には [SEP] を、結合したトークン列の先頭に [CLS] を末尾に [SEP] を加える。例を挙げると、「今日の天気は雨です。明日の天気は?」をトークン列にすると以下のようになる。

’[CLS]’, ’ 今日’, ’ の’, ’ 天気’, ’ は’, ’ 雨’, ’ です’, ’。’, ’[SEP]’, ’ 明日’, ’ の’, ’ 天気’, ’ は’, ’?’, ’[SEP]’

ここで、特殊トークン [SEP] は文章のペアの境界を示す役割や文章の列の末尾を示す役割がある。特殊トークン [CLS] は文章の先頭を示す役割がある。また、トークンは各単語の分散表現を表しているのに対して、[CLS] は文章の分散表現として用いることがで

きる。

文章をトークン列に変換した後は、それぞれのトークンをベクトルに変換して、BERT に入力する。その際は以下のような、トークン、文書のタイプ、文書中の位置それぞれに応じた 3 つのベクトルの和を BERT に入力する。

- Token Embedding：　各トークンの分散表現
- Segment Embedding：　各文書に対応する値、単一の文章の場合は 0 を、ペアの文章の場合は先頭の文章を 0、末尾の文章を 1 を入力する
- Position Embedding：　各文章の単語の位置を先頭を 0 とし入力する

2.4.2　事前学習

事前学習は、大規模な文書コーパスを用いて汎用的な言語のパターンを学習するために行われる。この時、使われる文書コーパスは、ラベルなしデータである。このラベルなしデータを用いて、BERT には、次の 2 つの学習を組み合わせて学習を行う。

マスク付き言語モデル

1 つ目の学習として、マスク付き言語モデルがある。BERT にある単語を周りの単語から予測するタスクを用いて学習を行う方法である。具体的には、ランダムに選ばれた 15% のトークンを [mask] という特殊トークンに変換する。そして、変換された文書を BERT に入力し、[mask] の位置に元々あったトークンを予測するというタスクである。

マスク付き言語モデルによる学習の具体例を示す。「今日の天気は雨だった。」という文のトークン列の「雨」を [MASK] に変換する。この時以下のようなトークン列となる。このトークン列が与えられたときに、[MASK] の部分が元々は「雨」であったと予測するように学習を行う。

’[CLS]’, ’ 今日’, ’ の’, ’ 天気’, ’ は’, ’ 雨’, ’ だっ’, ’ た’, ’。’, ’[SEP]’

Next Sentence Prediction

2 つ目の学習として、Next Sentence Prediction がある。BERT が二つの文の関係性を理解できるようにするタスクを用いて学習を行う方法である。学習時に、ペアの文が入力され、データの 50% のペアは二つ目の文が一つ目の文に連続する文であり、残りの 50% のペアにはランダムな文が選ばれる。そして、その入力された二つ目の文が連続したものであるか、そうでないかを判定するタスクを持いて学習を行う。

Next Sentence Prediction による学習の具体例を示す。「今日の天気は雨だ。」と「だから傘をさす。」という連続する 2 つの文があった時、ペアの文章の入力なので以下のようなトークン列となる。このトークン列が与えられたときに、2 つの文は連続している文章であると学習を行う。

. . . . . .

'[CLS]', ' 今日', ' の', ' 天気', ' は', ' 雨', ' だ', '。', '[SEP]', ' だから', ' 傘', ' を', ' さす', '。', '[SEP]'

. . . . . .

2.5　Sentence-BERT

Sentence-BERT [6] は、Reimers らが発表した BERT をファインチューニングして良質な文のベクトルを生成できるようにしたモデルである。Sentence-BERT の特徴は、単語の意味を捉えた分散表現を与える BERT とは違い、文の意味や文脈を考慮した文の分散表現を得ることができる点である。Sentence-BERT のファインチューニングは、ある文章とそれに類似した文章を学習データとして、似た文章同士は似た分散表現を得るように BERT を学習するという方法で行う。Sentence-BERT の構造を以下に示す。

２つの文章の類似性を、「含意」、「矛盾」、「どちらでもない」のような分類問題として学習する際は、左の図 2.1 のようにそれぞれのベクトルを連結して分類問題として解く。2 つの文章を数値として、類似度の度合いを学習する際は、右の図 2.2 のように２つのベクトルのコサイン類似度を比較する。

ここで、u と v は sentenceA と sentenceB のベクトルである。このベクトルは、sentenceA と sentenceB を BERT に入力し、ベクトルを固定長にするために出力を

pooling することで得る分散表現である。pooling では次の 3 つの方法が比較されている。

- MEAN：各トークンのベクトルの平均を取る
- MAX：各トークンのベクトルの最大値を取る
- CLS：[CLS] トークンのベクトルを取る

論文によると、この中では MEAN による pooling が最も良い精度であった。

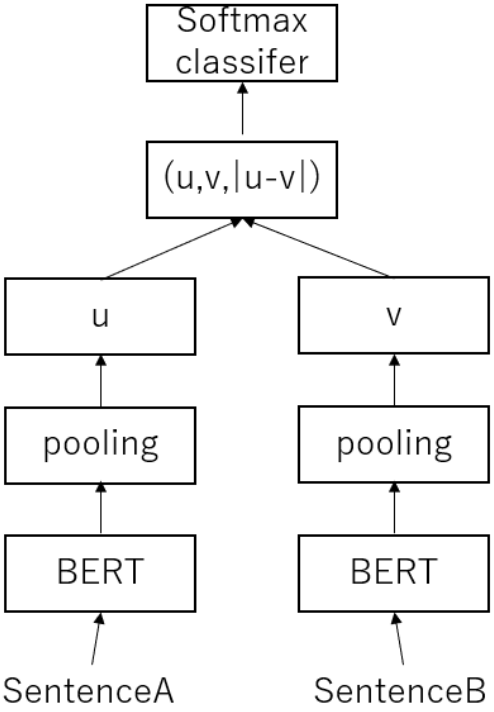

図 2.1: 分類問題として学習

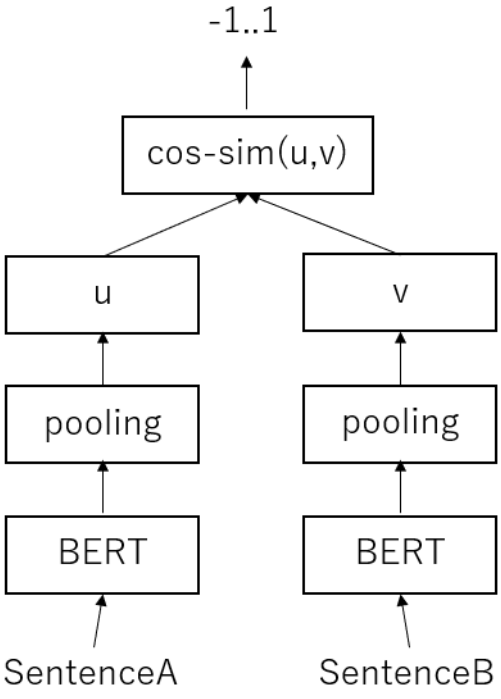

図 2.2: 類似度の度合いとして学習

2.6　ゼロショット学習

ゼロショット学習とは事前知識を利用することで、学習データに存在しないパターンの予測ができる機械学習手法である。ゼロショット学習は、学習データに存在しなかったラベルを含むすべてのラベルに対して、何らかの情報が既知であるという前提を置いているため、学習データに存在しないパターンを予測することが可能になる。

具体的には、ゼロショット学習の目標は Wang らの論文 [2] の通り、特徴量空間を X、正解ラベル空間を Y とし、学習データに存在したラベルのみによって表される空間を Y^{train}、学習データに存在しなかったラベルのみによって表される空間を Y^{target} とする

と、

$$f : X \to Y^{target} \tag{2.2}$$

となるような分類器 f を学習することが目的となる。

ただし、この時

$$Y^{train} \cup Y^{target} = Y \tag{2.3}$$

$$Y^{train} \cap Y^{target} = \emptyset \tag{2.4}$$

の両方が成り立つ。

第 3 章

ゼロショット学習による文書分類とラベル名選択の検討

ゼロショット学習による文書分類において、ラベル名と文章のベクトルの近さによる分類手法では、分類に不適切なラベル名のために分類精度が落ちる。本研究では、2 つの実験を行い、その実験結果を基に精度が落ちないラベル名の性質の検討を行う。実験の 1 つ目は、適当な独自ラベルを与えることで、ゼロショット学習による文書分類の精度が上がることを確認する。実験の 2 つ目は、3 つの手法で独自付与のラベルを与え分類精度を比較する。これらの結果を基に、ラベル名の性質を考える。

2 つの実験において各カテゴリにラベル名を与え、ゼロショット学習による文書分類で分類精度を確認するが、その際の分類の手順を以下に示す。

1. 各カテゴリに提案手法を用いてラベル名を与える
2. Sentence-BERT を用いて、ラベル名と文書を分散表現によりベクトル化する
3. 各カテゴリと文書とのコサイン類似度を計算する
4. コサイン類似度の値が大きい順に順位付けをする
5. 順位 1 位のカテゴリを分類カテゴリとする
6. データセットの文章数だけ上の手順を繰り返し、正しく分類できた文章数をカウントし精度を計算する

手順 4 に記述したコサイン類似度は以下の式で求まる。コサイン類似度は、2 つのベクトルの内積を、2 つのベクトルの大きさで割ることで計算される。この値は 2 つのベクト

ルのなす角度を表すため、ベクトル同士の近さを表す尺度となる。

$$cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|q| \cdot |d|} \tag{3.1}$$

3.1　独自付与によるラベル名

ゼロショット学習は、ラベル名の選択の方法が重要である。実験の 1 つ目である、独自付与によるラベル名の選択手法について述べる。独自付与によるラベルとは、直接各カテゴリの文章の内容を確認して、各カテゴリに所属する文章の意味を説明すると思われるラベル名を独自に考え与えることである。たとえば、livedoor ニュースコーパスには、「ムービーエンター」というカテゴリがある。このカテゴリのいくつかの文章を取り出し、内容を確認すると映画や俳優についての記事が多いと分かる。そこで、このカテゴリには「映画」や「俳優」といった単語をラベル名とする。

3.2　自動付与によるラベル名

実験の 2 つ目である、自動付与によるラベル名の選択について述べる。自動付与によるラベルとは、分類するデータセットからその文書を特徴づける単語を機械的に取り出し、ラベル名にする手法である。本実験では、Bag-of-Words、TF-IDF、中心の文章による自動付与ラベルを取り出す。

3.2.1　Bag-of-Words によるラベル名

Bag-of-Words を用いたラベル名の選択方法の手順を記述する。ただし、文書中のすべての単語を考慮したベクトルは次元数が膨大になるため，出現回数がある回数以上の単語のみをベクトルの次元に割り当てる。

1. Bag-of-Words を用いて、各文の単語の出現回数を数える
2. 各文を出現回数が多い単語順に順位付けをする
3. 各カテゴリごとに、カテゴリ内の各文の順位 1 位の単語をカウントする
4. 各カテゴリをカウントが多い単語順に順位付けをする
5. 各カテゴリごとに順位上位 k 個をラベル名の単語の候補とする

上記の手順でカテゴリ A のラベル名を、Bag-of-Words により選択する例を示す。

1. **各文の単語の出現回数を数える**

カテゴリ A のある文 d を Bag-of-Words によってベクトル化する。出現した単語を w、出現回数を c とする。

表 3.1: 文 d の Bag-of-Words によるベクトル

単語	w_1	...	w_n
出現回数	c_1	...	c_n

2. **出現回数順に順位付けをする**

d の Bag-of-Words によるベクトルを出現回数順に順位付けをする。この例では、出現回数順に並び替える。

表 3.2: 文 d の Bag-of-Words の順位付け

単語	$w_{argmax(c_1,...,c_n)}$	...	$w_{argmin(c_1,...,c_n)}$
出現回数	$c_{argmax(c_1,...,c_n)}$	...	$c_{argmin(c_1,...,c_n)}$

3. **各カテゴリごとに、カテゴリ内の文の順位 1 位の単語を数える**

手順 2 より、カテゴリ A 内の各文の順位 1 位の単語を求めた。各文の順位 1 位の単語の出現回数を数える。各文の 1 位に出現した単語を category_w、 出現回数を c とする。

表 3.3: カテゴリ内の順位 1 位の単語の出現回数

単語	$category\_w_1$	...	$category\_w_n$
出現回数	c_1	...	c_n

4,5. **各カテゴリごとに出現回数順に順位付けをし、上位 k 個をラベル名の候補とする**

手順 3 で得られた単語を出現回数順に順位付けを行い、上位 k 位をラベル名の候

補の単語とする。

表 3.4: カテゴリ内の順位 1 位の単語の順位付け

単語	$category\_w_{argmax(c_1,...,c_n)}$	...	$category\_w_{argmin(c_1,...,c_n)}$
出現回数	$c_{argmax(c_1,...,c_n)}$	...	$c_{argmin(c_1,...,c_n)}$

3.2.2　TF-IDF によるラベル名

TF-IDF を用いたラベル名の選択方法の手順を記述する。ただし、Bag-of-Words の時と同様、文書中のすべての単語を考慮したベクトルは次元数が膨大になるため，出現回数が一定以上の単語のみをベクトルの次元に割り当てる。

1. TF-IDF を用いて、各文の TF-IDF の値を計算する
2. 各文を TF-IDF の値が大きい単語順に順位付けする
3. 各カテゴリごとに、カテゴリ内の各文の順位 1 位の単語をカウントする
4. 各カテゴリをカウントが多い単語順に順位付けをする
5. 各カテゴリごとに順位上位 k 個をラベル名の単語の候補とする

TF-IDF によるラベル名選択の具体例は、Bag-of-Words によるラベル名選択と同様の操作が多いので省略する。異なる点は、手順 1 のベクトル化を TF-IDF を用いる点と、手順 2 の並び替えを TF-IDF の値で並び替える点である。

3.2.3　中心の文章によるラベル名

中心の文章を用いたラベル名の選択方法の手順を記述する。

1. Sentence-BERT を用いて各カテゴリの文をベクトル化する
2. 各カテゴリごとに、カテゴリ内の各文のベクトルを足して文章数で割ることで、カテゴリの平均のベクトルを計算する
3. 各カテゴリごとに、平均ベクトルにコサイン類似度が最も近い文書をラベル名とする

上記の手順であるカテゴリ A のラベル名を、中心の文章により選択する例を示す。

1. **カテゴリの各文をベクトル化する**

   カテゴリ $A = (d^1, d^2, ..., d^{|A|})$ の各文は、それぞれ Sentence-BERT の分散表現により $u^1, u^2, ..., u^{|A|}$ とベクトル化される

2. **カテゴリの平均のベクトルを求める**

   カテゴリ A の各ベクトル $(u^1, u^2, ..., u^{|A|})$ を足して、文章数 $|A|$ で割り平均のベクトル u_{mean} を求める。よって、計算式は次のようになる

   $u_{mean} = (u^1 + u^2 + ... + u^{|A|})/|A|$

3. **平均のベクトルに一番近い文をラベル名とする**

   u_{mean} と各ベクトル $u^1, u^2, ..., u^{|A|}$ をコサイン類似度を用いて測ることで、最も平均のベクトルに似ている文 u_{label} を見つけることができる

   $$u_{label} = \underset{l \in 1,2,...,|A|}{argmax}\ cossim(u_{mean}, u^l)$$

提案手法の全体の流れを図 3.1 に示す。

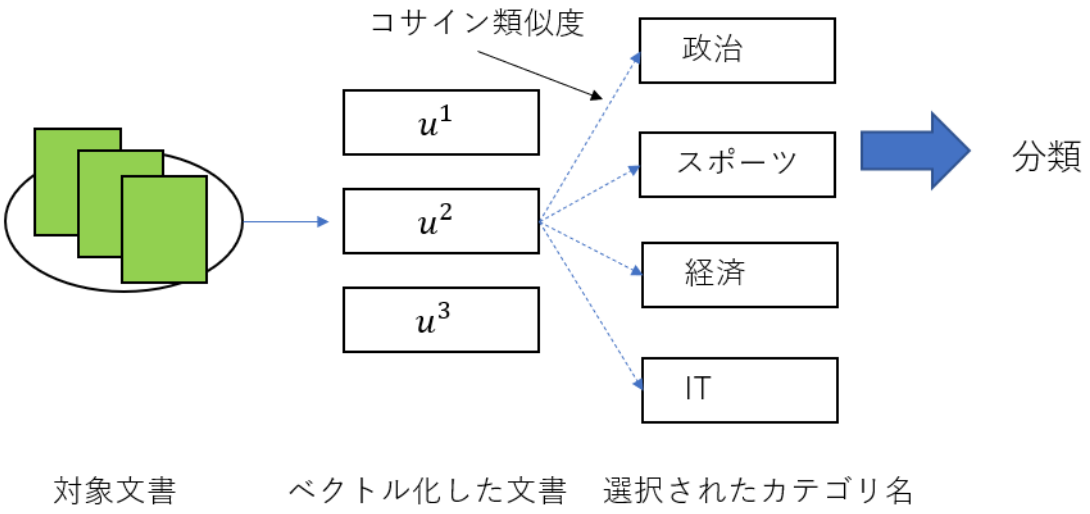

図 3.1: 提案手法全体の流れ

第 4 章

実験

4.1 事前学習済みモデル

本実験では、事前学習済みモデルとして HuggingFace で公開されている[*1]Sentence-BERT モデルを使用した。

このモデルは、Nils Reimersr らによる論文 [6] を基に作られた日本語版 Sentence-BERT モデルである。日本語版のモデルの作成において、huggingface/transformers（モデルは東北大学 乾・鈴木研究室様作）[*2]の日本語版 BERT モデルを用いている。

4.2 実験用データセット

本実験では、データセットとして livedoor ニュースコーパス[*3]を用いた。livedoor ニュースコーパスとは、NHN Japane 株式会社が運営する「livedoor ニュース」のうち、表 4.1 に示した 9 つのカテゴリに関するニュース記事を収集したコーパスである。

本実験には、訓練データは必要としないため、すべての文書を分類用データとして使用する。

[*1] https://huggingface.co/sonoisa/sentence-bert-base-ja-mean-tokens
[*2] https://github.com/huggingface/transformers
[*3] https://www.rondhuit.com/download.html

表 4.1: livedoor ニュースコーパスの文書内訳

ラベル名	文書数
独女通信	870
IT ライフハック	870
家電チャンネル	864
ライブドアホーム	511
ムービーエンター	870
ピーチィ	842
エスマックス	870
スポーツウォッチ	900
トピックニュース	770

4.3　実験方法

4.3.1　前処理

提案手法の Bag-of-Words と TF-IDF によるラベル名の自動付与では、候補となる単語を探す前に前処理を行う。Python のライブラリに neologdn[*4]がある。neologdn は、mecab-ipadic-NEologd の Wiki を基に正規化を行うライブラリである。本研究では、以下の前処理を行った。

- 英文字の小文字化
- 数字の削除
- neologdn を用いた文字のつづりや表記ゆれの吸収
- 名詞のみの取り出し

[*4] https://github.com/ikegami-yukino/neologdn

4.3.2　学習時の設定

Bag-of-Words の計算には、Python の scikit-learn ライブラリのクラスである CountVectorizer[*5]を用いる。このクラスは、文を入力したときに、各文の単語の出現頻度を自動でカウントを行うクラスである。この時、ある単語の出現する文章数がパラメータ以下の時その単語をカウントしない設定ができる。本研究ではそのパラメータの値を 0.15% とする。

TF-IDF の計算には、Python の scikit-learn ライブラリのクラスである TfidfVectorizer[*6]を用いる。このクラスは、文を入力したときに、各文の tf-idf 値を自動で計算する行うクラスである。この時、CountVectorizer と同様にある単語の出現する文章数がパラメータ以下の時その単語をカウントしない設定ができる。本研究ではそのパラメータの値を 0.15% とする。

4.3.3　ラベル名の付け方

Bag-of-Words と TF-IDF では、提案手法の通り各カテゴリから上位 k 個の単語を取り出すが、本実験では k=3 とする。すなわち、各カテゴリの頻度上位 3 つの単語を用いてラベル名とする。

また、ラベル名は単語として用いるのではなく短い文のラベル名とする。本実験では、ラベル名として与える文を「これは (選択した単語) についての文章です。」とする。そして、Sentence-BERT を使ってそれら 3 つ文章の平均のベクトルをラベル名とする。具体的には、以下 3 つの文のベクトルの平均をラベル名とする。

- 「これはスポーツについての文章です」
- 「これは選手についての文章です」
- 「これは試合についての文章です」

[*5] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
[*6] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

4.4　カテゴリに与えたラベル名

独自ラベル、Bag-of-Words、TF-IDF によりラベル名として選択した単語を表 4.2 に示す。表は、独自ラベル、Bag-of-Words、TF-IDF による手法において、各カテゴリから取り出した 3 つの単語を示している。表の各列は、オリジナルは元々 livedoor ニュースコーパスで与えられているラベル名であり表 4.1 に示したものである。独自が、独自付与により与えたラベル名、BOW が、Bag-of-Words により与えたラベル名、TFIDF が、TF-IDF により与えたラベル名である。

ここでは、単語のみを表に示したが、実際に用いたラベル名は、4.3.3 の通り「これは (選択した単語) に関する文章です。」としている。

表 4.2: 各カテゴリに与えたラベル名

オリジナル	独自	BOW	TFIDF
独女通信	女 男 結婚	人 歳 女性	結婚 女 歳
IT ライフハック	IT Google ソフトウェア	アプリ 孫社長 写真	孫社長 紺子 ロゴ
家電チャンネル	家電 電子 製品	話題 人 iphone	ビデオ salon hulu 撮影
ライブドアホーム	仕事 転職 年収	転職 ゴルフ 仕事	ゴルフ 転職 年収
ムービーエンター	映画 俳優 ドラマ	映画 公開 監督	映画 アベンジャーズ カーター
ピーチィ	美容 スイーツ ファション	女性 位 肌	当選 肌 お気に入り
エスマックス	スマートフォン パソコン モバイル	更新 アプリ ソフトウェア	更新 xperia アップデート
スポーツウォッチ	スポーツ 選手 試合	選手 試合 監督	ダルビッシュ 斎藤 試合
トピックニュース	芸能人 事件 政治	韓国 akb 日本	韓国 akb 橋本

4.5　実験結果

各手法を用いて選択したラベル名を用いて、ゼロショット学習による文書分類を行った。各カテゴリの正解率を、表 4.3 と図に示す。

baseline は、元々 livedoor ニュースコーパスのカテゴリに付けられていたラベル名をそのまま使用して文書分類を行った結果である。独自が、独自付与により与えたラベル名で実験を行った結果である。BOW が、Bag-of-Words により与えたラベル名で実験を行った結果である。TFIDF が、TF-IDF により与えたラベル名で実験を行った結果である。中心が中心の文章をラベル名として実験を行った結果である。表より、baseline のラベルよりも変更したすべてのラベル名で正解率が上がっている。また、この提案手法の中で、全体平均の正解率は、独自ラベルが一番高い結果となった。

表 4.3: 各カテゴリの正解率

オリジナルラベル名	baseline	独自	BOW	TFIDF	中心
独女通信	0.201	0.586	0.137	0.538	0.433
IT ライフハック	0.782	0.474	0.269	0.010	0.045
家電チャンネル	0.140	0.036	0.014	0.361	0.087
ライブドアホーム	0.086	0.240	0.323	0.130	0.052
ムービーエンター	0.123	0.750	0.589	0.052	0.348
ピーチィ	0.021	0.636	0.399	0.814	0.157
エスマックス	0.006	0.550	0.338	0.534	0.673
スポーツウォッチ	0.433	0.681	0.748	0.659	0.468
トピックニュース	0.086	0.196	0.069	0.036	0.106
全体平均	0.216	0.462	0.339	0.363	0.282

第 5 章

考察

5.1 実験結果に関する考察

実験結果から、独自付与と自動付与のいずれでもラベル名を変更することで、平均の正解率が向上したため、ラベル名変更の有効性が確認された。特に、独自付与によるラベル名は、baseline のラベル名よりも 0.2 以上、自動付与のラベル名よりも 0.1 以上、正解率が改善した。

baseline が他の手法と比べて精度が低い理由は、「ライブドアホーム」、「ピーチィ」、「エスマックス」等が、各カテゴリの文の内容に沿ったラベル名でなかったためと考えられる。

自動付与によるラベル名が、独自付与によるラベル名よりも精度が低い理由を考える。表 4.2 から、BOW や TF-IDF は、ラベル名として選択した単語に固有名詞が独自付与よりも多く選択されている。各カテゴリの文は、同じカテゴリ内でも話題の違いによる影響で様々な分布を持つため、ラベル名から得られる特徴と文の特徴が異なる。したがって、自動付与によるラベル名の固有名詞は一部の文にはよく適合するが、他の文には全く適合しないために精度が低くなったと考えられる。一方で、独自付与によるラベル名は、固有名詞ではなく一般名詞が多いためより多くの文に適合し精度が良くなったと考えられる。

中心の文章の正解率についての考察は、次の節で述べる。

5.2　ラベル名の性質に関する考察

ゼロショット学習による文書分類の精度が上がるラベル名について考察する。前節の考察でも書いたように、カテゴリ内の多くの話題に適合する単語がより良い。したがって、固有名詞よりも一般名詞の方が良いと考えられる。また、「独女通信」において独自が「結婚・女・男」を選択したことに対し BOW は「結婚・女・人」を選択した。結果は表 4.2 の通り、独自が 0.586、BOW が 0.137 であった。これは、「人」が「男」よりも広い概念であることが原因の一つであると考えられる。このことは、上位語下位語の関係に帰着できると考える。固有名詞は下位語であるが、これは一部の話題にしか適合しない。対して、あまり上位の単語もカテゴリの意味を説明できているとは言えず、結果どの文にも適合しなくなってしまう。ただ現状、どの階層の上位語下位語が良いラベル名となるかの研究は行っていない。

しかし、「ライブドアホーム」や「トピックニュース」の結果から、同じカテゴリ内に多くの話題含まれている場合、3 単語程度を使用したラベル名では、いくら一般化したラベル名でもそのラベル名から外れた内容の文が存在してしまい分類精度は頭打ちになってしまうと考えられる。つまり、分類する文書自体が各カテゴリにうまく分けられていない場合、ゼロショット学習のみで精度を上げることは難しい。解決方法としては、3 単語以上の単語を使うラベル名とする、分類するカテゴリの数を増やすなどが考えられる。カテゴリの文の話題が広すぎる事は、中心の文章の結果が他の自動付与の手法と比べ悪かったことにつながる。具体的には、話題が多いと文の分布が広いため、中心の文章から離れた文が多くなってしまい結果が悪くなったと考えられる。

第 6 章

結論

本研究では、独自付与のラベル、自動付与のラベルとして Bag-of-Words・TF-IDF・中心の文章を用いて、ゼロショット学習による文書分類の精度が上がるラベル名の考察を行った。実験では、Sentence-BERT の分散表現を用いて、コサイン類似度により近さを測ることで分類を行ったところ、もともと与えられていたラベル名よりも、独自ラベル・自動付与ラベルの全てでゼロショット学習による文書分類の精度が上がる結果となった。この結果から、ゼロショット学習による文書分類においてラベル名の変更は有効であり、精度が上がるラベル名とは、各カテゴリの意味を説明する上位語下位語を考慮した一般名詞が分類の精度が上がるラベル名であると考察を行った。

今後の課題としては、考察でも述べた通り上位語下位語の概念を考慮したラベル名の分析が必要である。具体的には、自動付与で選択された単語を上位下位に階層的に選択していくことで行うことができる。また、本研究では、ラベル名を分類する文章から直接自動付与により与えていたが、これでは真の意味でゼロショット学習とは言えない。そのため、この研究の結果を基に別の文章集合からラベル名を得る必要がある。

謝辞

本研究を進めるにあたって，多くのご指導を頂いた指導教員の新納浩幸教授に感謝申し上げます．また，日々の活動を通し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] Prateek Veeranna Sappadla, Jinseok Nam, Eneldo Loza Mencía, and Johannes Fürnkranz. Using semantic similarity for multi-label zero-shot classification of text documents. In *The European Symposium on Artificial Neural Networks*, 2016.

[2] Wei Wang, Vincent W. Zheng, Han Yu, and Chunyan Miao. A survey of zero-shot learning: Settings, methods, and applications. *ACM Trans. Intell. Syst. Technol.*, Vol. 10, No. 2, jan 2019.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, Vol. abs/1301.3781, , 2013.

[4] 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, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

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

[6] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, Vol. abs/1908.10084, , 2019.

付録

A　本実験で使用したプログラム

本実験で使用した、Bag-of-Words によるラベル名選択のプログラムを A.1 に、TF-IDF によるラベル名選択のプログラムを A.2 に、中心の文章によるラベル名選択のプログラムを A.3 に、ゼロショット学習による文書分類のソースコードを A.4 に示す。

ソースコード A.1: bag-of-words.py

```
import numpy as np
import MeCab
import re
import neologdn
import glob
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer

#データセットの作成
category_list = [
    'dokujo-tsushin',
    'it-life-hack',
    'kaden-channel',
    'livedoor-homme',
    'movie-enter',
    'peachy',
    'smax',
    'sports-watch',
    'topic-news'
]
texts = []
category_sentence_num = []
cnt = 0
```

```
24
25 for label, category in enumerate(tqdm(category_list)):
26     category_sentence_num.append(cnt)
27     for file in glob.glob(f'./text/{category}/{category}*'):
28         cnt += 1
29         lines = open(file).read().splitlines()
30         text = '\n'.join(lines[3:]) # ファイルの行目から文書を抜き出す。
               4
31         text = text.replace("\n", "")
32         texts.append(text)
33
34 #前処理
35 for i in range(len(texts)):
36     texts[i] = texts[i].lower() #小文字化
37     texts[i] = re.sub(r'\d+', '', texts[i]) #数値変換
38     texts[i] = neologdn.normalize(texts[i]) #正規化
39
40 #ストップワード処理
41 path = "-d␣/usr/lib/x86_64-linux-gnu/mecab/dic/mecab-ipadic-neologd"
42 tagger = MeCab.Tagger(path)
43 for i in range(len(texts)):
44     words = tagger.parse(texts[i]).split('\n')
45     res = []
46     for j in range(len(words)):
47         words[j] = words[j].replace('\t',',').split(',')
48         if words[j][0] == 'EOS':
49             break
50         if words[j][1] in ['名詞']:
51             res.append(words[j][0])
52     texts[i] = '␣'.join(res)
53
54 vectorizer = CountVectorizer(min_df=0.0015, analyzer=lambda x: x) #よ
       り小さい単語は削る 0.0015
55 word2vec = vectorizer.fit_transform(texts).toarray()
56
57 index = word2vec.argsort(axis=1)[:,::-1] #各文を Bag-of-の値が大きい単
       語のインデックス順に並び替える Words
58 feature_names = np.array(vectorizer.get_feature_names()) #
       get_feature_names()を化
       numpy
59 feature_words = feature_names[index]
60
61 #各カテゴリごとに、カテゴリ内の各文の順位１位の単語をカウントする
```

```
candidate_word = list({} for _ in range(9)) #各ラベルのキーワード
which_label = 0 #今扱っている文がどのカテゴリであるかを判断する
for i in range(len(texts)):
    if which_label+1 < 9 and i > labels_num[which_label+1]:
        which_label += 1

    for j in range(1):
        if feature_words[i][j] not in candidate_word[which_label]:
            candidate_word[which_label][feature_words[i][j]] = 1
        else:
            d[which_label][feature_words[i][j]] += 1

for i in range(9):
    d[i] = sorted(d[i].items(), key=lambda x:x[1], reverse=True) #各
        カテゴリごとにカウントが多い単語順に並び替える
```

ソースコード A.2: tfidf.py

```
import numpy as np
import MeCab
import re
import neologdn
import glob
from tqdm import tqdm
from sklearn.feature_extraction.text import TfidfVectorizer

#データセットの作成
category_list = [
    'dokujo-tsushin',
    'it-life-hack',
    'kaden-channel',
    'livedoor-homme',
    'movie-enter',
    'peachy',
    'smax',
    'sports-watch',
    'topic-news'
]
texts = []
category_sentence_num = []
cnt = 0

```

```
for label, category in enumerate(tqdm(category_list)):
    category_sentence_num.append(cnt)
    for file in glob.glob(f'./text/{category}/{category}*'):
        cnt += 1
        lines = open(file).read().splitlines()
        text = '\n'.join(lines[3:]) # ファイルの行目から文書を抜き出す。
            4
        text = text.replace("\n", "")
        texts.append(text)

#前処理
for i in range(len(texts)):
    texts[i] = texts[i].lower() #小文字化
    texts[i] = re.sub(r'\d+', '', texts[i]) #数値変換
    texts[i] = neologdn.normalize(texts[i]) #正規化

#ストップワード処理
path = "-d␣/usr/lib/x86_64-linux-gnu/mecab/dic/mecab-ipadic-neologd"
tagger = MeCab.Tagger(path)
for i in range(len(texts)):
    words = tagger.parse(texts[i]).split('\n')
    res = []
    for j in range(len(words)):
        words[j] = words[j].replace('\t',',').split(',')
        if words[j][0] == 'EOS':
            break
        if words[j][1] in ['名詞']:
            res.append(words[j][0])
    texts[i] = '␣'.join(res)

#によりラベル名を選択
vectrizer = TfidfVectorizer(min_df=0.0015, token_pattern='(?u)\\b\\w
    +\\b') #より小さい単語は削る
    0.0015
vectrizer.fit(texts)
tfidf = vectrizer.transform(texts).toarray()

index = tfidf.argsort(axis=1)[:,::-1] #各文を TF-の値が大きい単語のイン
    デックス順に並び替える IDF
feature_names = np.array(vectrizer.get_feature_names()) #
    get_feature_names()を化
    numpy
feature_words = feature_names[index]
```

```
62
63 #各カテゴリごとに、カテゴリ内の各文の順位１位の単語をカウントする
64 candidate word = list({} for _ in range(9)) #各ラベルのキーワード
65 which_label = 0 #今扱っている文がどのカテゴリであるかを判断する
66 for i in range(len(texts)):
67     if which_label+1 < 9 and i > labels_num[which_label+1]:
68         which_label += 1
69
70     for j in range(1):
71         if feature_words[i][j] not in candidate word[which_label]:
72             candidate word[which_label][feature_words[i][j]] = 1
73         else:
74             d[which_label][feature_words[i][j]] += 1
75
76 for i in range(9):
77     d[i] = sorted(d[i].items(), key=lambda x:x[1], reverse=True) #各
       カテゴリごとにカウントが多い単語順に並び替える
```

ソースコード A.3: center.py

```
1 import numpy as np
2 import MeCab
3 import re
4 import neologdn
5 import glob
6 from tqdm import tqdm
7
8 #データセットの作成
9 category_list = [
10     'dokujo-tsushin',
11     'it-life-hack',
12     'kaden-channel',
13     'livedoor-homme',
14     'movie-enter',
15     'peachy',
16     'smax',
17     'sports-watch',
18     'topic-news'
19 ]
20 texts = []
21 category_sentence_num = []
22 cnt = 0
```

```
23
24 for label, category in enumerate(tqdm(category_list)):
25     category_sentence_num.append(cnt)
26     for file in glob.glob(f'./text/{category}/{category}*'):
27         cnt += 1
28         lines = open(file).read().splitlines()
29         text = '\n'.join(lines[3:]) # ファイルの行目から文書を抜き出す。
               4
30         text = text.replace("\n", "")
31         texts.append(text)
32
33 #前処理
34 for i in range(len(texts)):
35     texts[i] = texts[i].lower() #小文字化
36     texts[i] = re.sub(r'\d+', '', texts[i]) #数値変換
37     texts[i] = neologdn.normalize(texts[i]) #正規化
38
39 #ストップワード処理
40 path = "-d␣/usr/lib/x86_64-linux-gnu/mecab/dic/mecab-ipadic-neologd"
41 tagger = MeCab.Tagger(path)
42 for i in range(len(texts)):
43     words = tagger.parse(texts[i]).split('\n')
44     res = []
45     for j in range(len(words)):
46         words[j] = words[j].replace('\t',',').split(',')
47         if words[j][0] == 'EOS':
48             break
49         if words[j][1] in ['名詞']:
50             res.append(words[j][0])
51     texts[i] = '␣'.join(res)
52
53 which_label = 0 #今扱っている文がどのカテゴリであるかを判断する
54 category_sentence_num = [0]*9 #各ラベルの文書の数
55 label_average = [0]*9 #各カテゴリの中心のベクトル
56
57 for i, sentence in enumerate(texts):
58     if which_label+1 < 9 and i > labels_num[which_label+1]:
59         which_label += 1
60
61     category_sentence_num[which_label] += 1
62     label_average[which_label] += sentence
```

```
63
64 for i in range(9):
65     label_average[i] /= category_sentence_num[i]
66
67 #中心のベクトルに最も近い分の探索
68 most_similar_label = [] #各カテゴリの最も近いタイトル
69
70 for i in range(9):
71     label_tmp = '' #一時的に最も近い文を格納
72     max_cos_similarity = 0 #各カテゴリの最大コサイン類似度
73
74     for j in range(category_sentence_num[i]):
75         #各文と中心のベクトルのコサイン類似度を測る
76         nr1 = np.linalg.norm(label_average[i], ord=2)
77         nr2 = np.linalg.norm(texts[j], ord=2)
78         cos_similarity_tmp = np.dot(label_average[i], texts[j]) / (nr1
               * nr2)
79
80         if j == 0: #最初の文章ならば必ず最も近い文なので格納
81             max_cos_similarity = cos_similarity_tmp
82             label_tmp = texts[j]
83         else: #それ以降はいままで近い文とのコサイン類似度の比較
84             if cos_similarity_tmp > max_cos_similarity:
85                 max_cos_similarity = cos_similarity_tmp
86                 label_tmp = texts[j]
87
88     most_similar_label.append(label_tmp)
```

ソースコード A.4: zero-shot-classification.py

```
1 from transformers import BertJapaneseTokenizer, BertModel
2 import torch
3 import numpy as np
4 import glob
5 from tqdm import tqdm
6
7 class SentenceBertJapanese:
8     def __init__(self, model_name_or_path, device=None):
9         self.tokenizer = BertJapaneseTokenizer.from_pretrained(
              model_name_or_path)
10        self.model = BertModel.from_pretrained(model_name_or_path)
```

```
11          self.model.eval()
12
13          if device is None:
14              device = "cuda" if torch.cuda.is_available() else "cpu"
15          self.device = torch.device(device)
16          self.model.to(device)
17
18      def _mean_pooling(self, model_output, attention_mask):
19          token_embeddings = model_output[0] #First element of
                model_output contains all token embeddings
20          input_mask_expanded = attention_mask.unsqueeze(-1).expand(
                token_embeddings.size()).float()
21          return torch.sum(token_embeddings * input_mask_expanded, 1) /
                torch.clamp(input_mask_expanded.sum(1), min=1e-9)
22
23      @torch.no_grad()
24      def encode(self, sentences, batch_size=8):
25          all_embeddings = []
26          iterator = range(0, len(sentences), batch_size)
27          for batch_idx in iterator:
28              batch = sentences[batch_idx:batch_idx + batch_size]
29
30              encoded_input = self.tokenizer.batch_encode_plus(batch,
                    padding="longest",
31                                                  truncation=True,
                                                        return_tensors="pt").to(
                                                        self.device)
32              model_output = self.model(**encoded_input)
33              sentence_embeddings = self._mean_pooling(model_output,
                    encoded_input["attention_mask"]).to('cpu')
34
35              all_embeddings.extend(sentence_embeddings)
36
37          # return torch.stack(all_embeddings).numpy()
38          return torch.stack(all_embeddings)
39
40  #Sentence-BERT
41  model_sentence = SentenceBertJapanese("sonoisa/sentence-bert-base-ja-
        mean-tokens")
42
43  #データセットの作成
```

```
category_list = [
    'dokujo-tsushin',
    'it-life-hack',
    'kaden-channel',
    'livedoor-homme',
    'movie-enter',
    'peachy',
    'smax',
    'sports-watch',
    'topic-news'
]
texts = []
category_sentence_num = []
cnt = 0

for label, category in enumerate(tqdm(category_list)):
    category_sentence_num.append(cnt)
    for file in glob.glob(f'./text/{category}/{category}*'):
        cnt += 1
        lines = open(file).read().splitlines()
        text = '\n'.join(lines[3:]) # ファイルの行目から文書を抜き出す。
            4
        texts.append(text)

#文書の分散表現
texts = model_sentence.encode(texts)

#ラベル名の作成
labels = []

for i in range(9):
    for j in range(3):
        labels.append('これは'+candidate word[i][j][0]+'についての文章
            です')
labels = model_sentence.encode(labels)

# ラベル名の平均を取る
labels_average = []

for i in range(9):
    ave = 0
    for j in range(n):
```

```
84          ave += labels[i*n+j]/n
85      labels_average.append(ave)
86
87  #ゼロショット学習による文書分類
88  which_label = 0 #今扱っている文がどのカテゴリであるかを判断する
89  correct_label = [0]*9
90
91  for i, v1 in enumerate(texts):
92      if which_label+1 < 9 and i > category_sentence_num[which_label+1]:
93          which_label += 1
94
95      cosine_similarity = []
96
97      #各ラベル名と文のコサイン類似度を求める
98      for j in range(9):
99          nr1 = np.linalg.norm(v1, ord=2)
100         nr2 = np.linalg.norm(labels_average[j], ord=2)
101         cosine_similarity.append(np.dot(v1, labels_average[j]) / (nr1
                * nr2))
102
103     max_value = max(cosine_similarity)
104     max_index = cosine_similarity.index(max_value) #コサイン類似度が最
            大のカテゴリを取り出す
105     if max_index == which_label: #今扱っているカテゴリととコサイン類似度
            が最大のカテゴリが等しいならば正解
106         correct_label[which_label] += 1
107
108 print('accuracy␣:␣' + str(sum(correct_label)/sum(category_sentence_num
        ))) #正解率の表
        示
```