language_culc

word2vecを用いた深層学習

Myullos

2025年3月3日

1 .概要

今回のプログラムはWikipediaのdumpデータを用いることによって、深層学習を行いcos類似度とベクトルを用いることによって、言語を数値化し単語の成分の抽出および単語同士の関連性、合成による単語同士の加法や減法の実現を試みる。

2.作成方法

①言語データを入手する。

Ex)wikipedia dumpデータ、the brown corpus、Amazon review corpus

また,言語データをwikiextractorを用いて解凍し、一つのテキストファイルに結合する。

②Mecabを用いることで形態素解析（文章を単語に分ける行為）を行う

③Word2Vecを用いて学習する。

④生成されたデータセットを使用する。

2-1①の詳細

①は、1.Wikipediaのダンプデータを入手

2.WikiExtractorのインストールと使用

3.抽出したデータを1つのファイルに結合

を行う。

1. Wikipediaのダンプデータを入手

手順:

Wikipedia Dumpsのサイト（https://dumps.wikimedia.org/）にアクセス。

必要な言語のWikipediaダンプを選択（日本語版なら /jawiki/）。

（pages-articles.xml.bz2 ）というファイルを探してダウンロード。

これは記事本文のみを含む圧縮ファイルで、通常（ jawiki-latest-pages-articles.xml.bz2） のような名前である。

```
Wikipedia Dumps
│
├─── jawiki/
│    ├─── 20240301/ （日付ごとのフォルダ）
│    │    ├─── jawiki-20240301-pages-articles.xml.bz2 （記事データ）
│    │    ├─── jawiki-20240301-pages-meta-current.xml.bz2 （すべてのリビジョン）
│    │    ├─── ...
│    │
│    ├─── latest/ （最新のダンプ）
│    │    ├─── jawiki-latest-pages-articles.xml.bz2
│    │    ├─── ...
```

↑このような構造。

2. WikiExtractorのインストールと使用

WikiExtractorは、WikipediaのXMLダンプをプレーンテキストに変換するPythonツール。

※Python（3.x）がインストールされていることを前提。

以下のコマンドでWikiExtractorをクローンする：

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

```
git clone https://github.com/attardi/wikiextractor.git
cd wikiextractor
pip install -r requirements.txt
```

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

この後、ダンプデータを抽出する

ダウンロードした（ jawiki-latest-pages-articles.xml.bz2 ）を展開：

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

```
python WikiExtractor.py -b 500M -o extracted jawiki-latest-pages-articles.xml.bz2
```

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

上記コマンドのオプション説明

-b 500M : 出力ファイルを 500MB ごとに分割

-o extracted : 出力フォルダ（extracted/）を指定

抽出後、フォルダ extracted/ 内にプレーンテキストデータが複数のファイルに分割されて保存される。

```
extracted/
 |
 ├── AA/
 |    ├── wiki_00
 |    ├── wiki_01
 |    ├── wiki_02
 |    ├── …
 |
 ├── AB/
 |    ├── wiki_00
 |    ├── wiki_01
 |    ├── wiki_02
 |    ├── …
 |
…
```

↑上図のような構造。

3. 抽出したデータを1つのファイルに結合

以下のコマンドで全ファイルを1つにまとめる：

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

```
cat extracted/*/wiki_* > wikipedia_text.txt
```

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

これで、wikipedia_text.txt にすべてのデータが統合される。

まとめ

1 .Wikipediaのダンプデータを取得

Wikipedia Dumps から pages-articles.xml.bz2 をダウンロード

2.WikiExtractorを使って抽出

python WikiExtractor.py -b 500M -o extracted
jawiki-latest-pages-articles.xml.bz2 を実行

3.複数ファイルを結合

cat extracted/*/wiki_* > wikipedia_text.txt を実行

2-2②の詳細

Wikipediaのテキストデータ（wikipedia_text.txt）をword2vecで学習させるために、以下の手順で前処理を行う。

1.不要な記号の削除（クリーニング）

2.MeCabによる形態素解析（単語分割・ストップワード除去)

1. 不要な記号の削除（クリーニング）

Wikipediaのテキストには、記号・数字・HTMLタグなどの不要な情報が含まれている。

これを除去し、word2vecの学習に適したテキストデータを作成する。

(1) クリーニングの手順

（）[]{}<>!?"' などの記号を削除（半角,全角）

改行・空白の統一を行う。

ーー

```
import re
with open("wikipedia_text.txt", 'r', encoding='UTF-8') as f:
    data = f.read()
data = re.sub(r"<.*?>", r"", data)
half_width_symbols = r'[!"#$%&\'\\()*+,-./:;<=>?@[\\]^_`{|}~「」〔〕“”〈〉『』
【】＆＊・（）＄＃＠。、？！｀＋￥％]' ＃半角記号の削除

data = re.sub(half_width_symbols, '', data)
full_width_symbols = r"[\uFF01-\uFF0F\uFF1A-\uFF20\uFF3B-\uFF40\uFF5B-
\uFF65\u3000-\u303F]" ＃全角記号の削除

data = re.sub(full_width_symbols, '', data)
output_file = 'clean_wikipedia_text.txt' ＃アウトプットのパスを指定

with open(output_file, 'w', encoding='UTF-8') as wf:
    wf.write(data)
print("記号削除が完了しました。結果は 'clean_wikipedia_text.txt' に保存されました。")
```

ーー

↑上記のプログラミング（今回はファイル内のremove.pyを使用した）
図解：クリーニング前後
ーー

【元データ】
東京都（とうきょうと、英: Tokyo）は、日本の首都である。[[23区]]と多摩地域からなる。
東京都の面積は2,194 km²であり、人口は約1,400万人（2024年時点）である。

【クリーニング後】
東京都 とうきょうと は 日本の首都である と多摩地域からなる
東京都の面積は であり 人口は 約 万人である

ーー

2. MeCabによる形態素解析

※形態素解析とは？

形態素解析（けいたいそかいせき）とは、日本語の文章を 単語（形態素）ごとに分解 する処理のこと。

日本語は英語のように単語と単語の間に スペースがない ため、単語の区切りを判別する必要がある。

例えば、「私は学校に行きます。」という文を形態素解析すると、次のように分解される。

私 / は / 学校 / に / 行き / ます / 。

このように単語を識別することで、機械が日本語を理解しやすくなる。

形態素解析を行い、テキストを単語ごとに分割する。

word2vecでは、単語単位での学習が必要なので、この処理が重。

(1) MeCabのインストール

まず、MeCabと辞書をインストールする。

————————————————————————————————————

```
pip install mecab-python3
```

————————————————————————————————————

↑上記のプログラム

その後、Pythonで形態素解析を実行する。

————————————————————————————————————

```
import MeCab
# MeCabの初期化（neologd辞書を使う場合は辞書パスを指定）
mecab = MeCab.Tagger('-Owakati')
# 形態素解析の関数
def tokenize_text(text):
    return mecab.parse(text).strip()
# クリーンなテキストデータを形態素解析
input_file = "clean_wikipedia_text.txt"
output_file = "tokenized_wikipedia_text.txt"
with open(input_file, "r", encoding="utf-8") as infile, open(output_file, "w",
encoding="utf-8") as outfile:
```

```
    for line in infile:
        tokenized_line = tokenize_text(line)
        if tokenized_line:
            outfile.write(tokenized_line + "\n")
```
ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

↑上記のプログラム

[ポイント]

-Owakati オプションを使用 → 分かち書き形式で出力（単語ごとにスペース区切り）
空白行を削除し、整形

図解：形態素解析の前後
ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

【元データ（クリーニング済み）】

東京都 とうきょうと は 日本の首都である と多摩地域からなる

【形態素解析後】

東京都 とうきょうと は 日本 の 首都 で ある と 多摩 地域 から なる
ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

このように単語がスペースで区切られるため、word2vecに適した形式になる。

まとめ

1.不要な記号を削除（クリーニング）

記号・数字・英字・URL・空白を除去し、クリーンなテキストを作成。

2.MeCabで形態素解析（分かち書き）

- Owakati で単語ごとにスペースごとに区切る。

2-3③の詳細

word2vecとは？

Word2Vecは、単語を数値（ベクトル）で表現する技術。
この技術を使うと、単語の意味の違いを機械が数値で理解できるようになる。

1. なぜWord2Vecが必要なのか？

コンピュータは 数字の計算は得意 ですが、言葉の意味を直接理解することはできない。
例えば、次のような文章があったとする。
「私は学校に行きます。」
このままではコンピュータは 「私は」「学校」「行きます」 という単語の意味を理解できない。
そこで、単語を数値（ベクトル）として表現する必要がある。

2.word2vecが単語をベクトルにする原理は？

Word2Vecの基本的なアイデアは、単語の意味は周囲の単語（コンテキスト）によって決まる という考え方に基づいている。
つまり、「似た意味を持つ単語は、似たような文脈で使われる」 という原則を利用して、単語のベクトルを学習する。
例：

1. 猫はかわいい。
2. 犬はかわいい。

→ 「猫」と「犬」は「かわいい」という単語と一緒に登場するため、意味が近いと推測できる。
このように、単語の前後にある単語を利用して、単語を数値（ベクトル）に変換していく。

3 . 単語をどのように数値化するのか？

単語をベクトルにする最も単純な方法として、**one-hot encoding**（ワンホットエンコーディング） がある。
例えば、次のような単語のリストがあるとします。

単語	One-hot ベクトル
猫	[1, 0, 0, 0, 0]
犬	[0, 1, 0, 0, 0]
りんご	[0, 0, 1, 0, 0]
車	[0, 0, 0, 1, 0]
花	[0, 0, 0, 0, 1]

[問題点]

単語が増えると ベクトルが非常に大きくなる（数百万次元になる可能性がある）

上記より、単語同士の意味の関係が分からない（「猫」と「犬」は似ているが、one-hotでは関係性が分からない）

この問題点を解決するためにword2vecを用いる。

４.Word2Vecによる単語のベクトル化

Word2Vecでは、one-hotのような単純な方法ではなく、学習によって意味のあるベクトルを作る仕組みになっている。

Word2Vecは、以下のような学習方法を使って単語ベクトルを作成する。

① CBOW（Continuous Bag of Words）

CBOW（シーボウ）は、周囲の単語からターゲットの単語を予測する方法。
例：

「私は 学校 に 行きます。」

↑「私は」と「に」「行きます」の3つの単語を使って、「学校」を予測します。

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

（私は, に, 行きます） → 学校

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

CBOWの特徴

高速に学習できる
よく使われる単語のベクトルをうまく学習できる

② Skip-gram

Skip-gramは、ターゲットの単語から周囲の単語を予測する方法 です。

「私は 学校 に 行きます。」

↑「学校」という単語を使って、前後の単語を予測します。

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

学校 → （私は, に, 行きます）

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

Skip-gramの特徴

データが少なくても学習できる
低頻度の単語に強い

5.Word2Vecがどのように単語ベクトルを学習する？

Word2Vecでは、ニューラルネットワークを使って単語のベクトルを学習する。
また、学習手順は以下の流れである。

5-1入力層（one-hotベクトル）

最初に、one-hotベクトルを入力として用意する。

例えば、単語リストが「猫, 犬, りんご, 車, 花」の5つだけだと仮定した時、

猫 → [1, 0, 0, 0, 0]

というone-hotベクトルが入力になる。

5-2隠れ層（重み行列W）

入力されたベクトルは、学習するための「重み行列（W）」を通じて変換される。

また、重み行列 W は、各単語を低次元のベクトルに変換する役割を持つ。

例えば、5単語 × 3次元の重み行列 W があるとすると、

	次元1	次元2	次元3
猫	0.12	-0.98	0.45
犬	0.15	-1.02	0.43
りんご	-0.75	0.88	-0.12
車.	0.33.	-0.24	0.57
花.	-0.52.	0.47	-0.34

※あくまで、重みは例であり計算過程はもう少し複雑。

また、このようにして各単語が 数値のベクトルで表現 されるようになる。

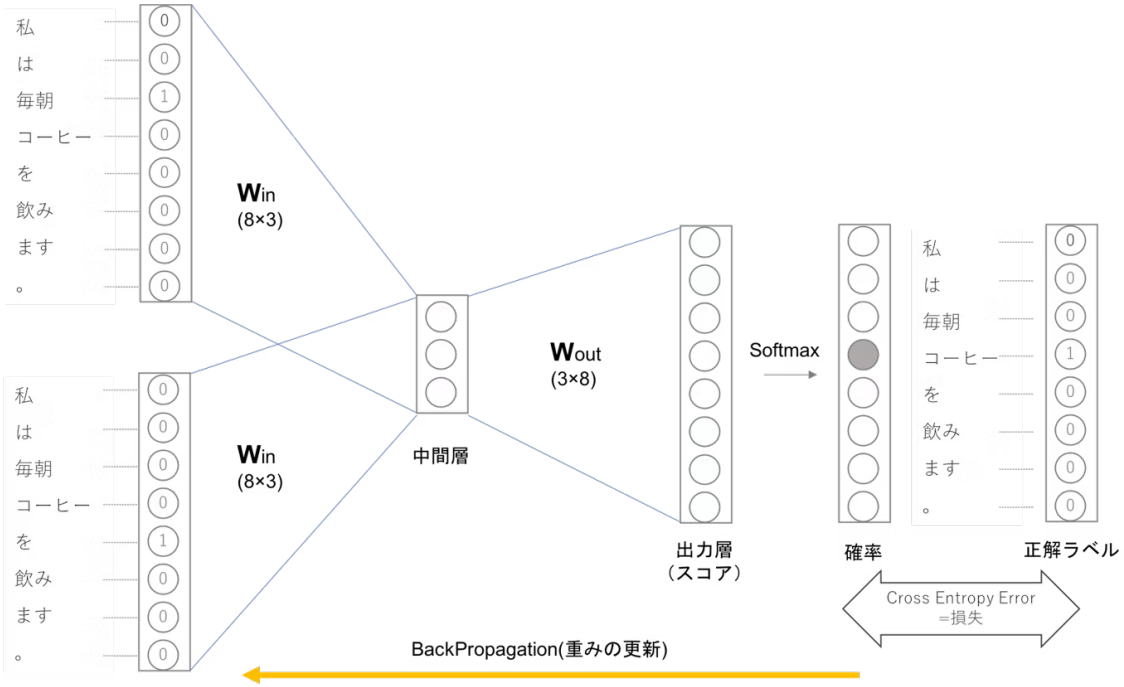
5-3出力層（予測）

CBOWなら、このベクトルを使ってターゲット単語を予測する。 また、Skip-gramなら、ターゲット単語から周りの単語を予測する。

この学習を繰り返すことで、重み行列 W の値が適切に調整され、単語の意味を反映したベクトルになっていく。

CBOWの図

また、損失関数（コンテクストを前後 1 単語にした場合）

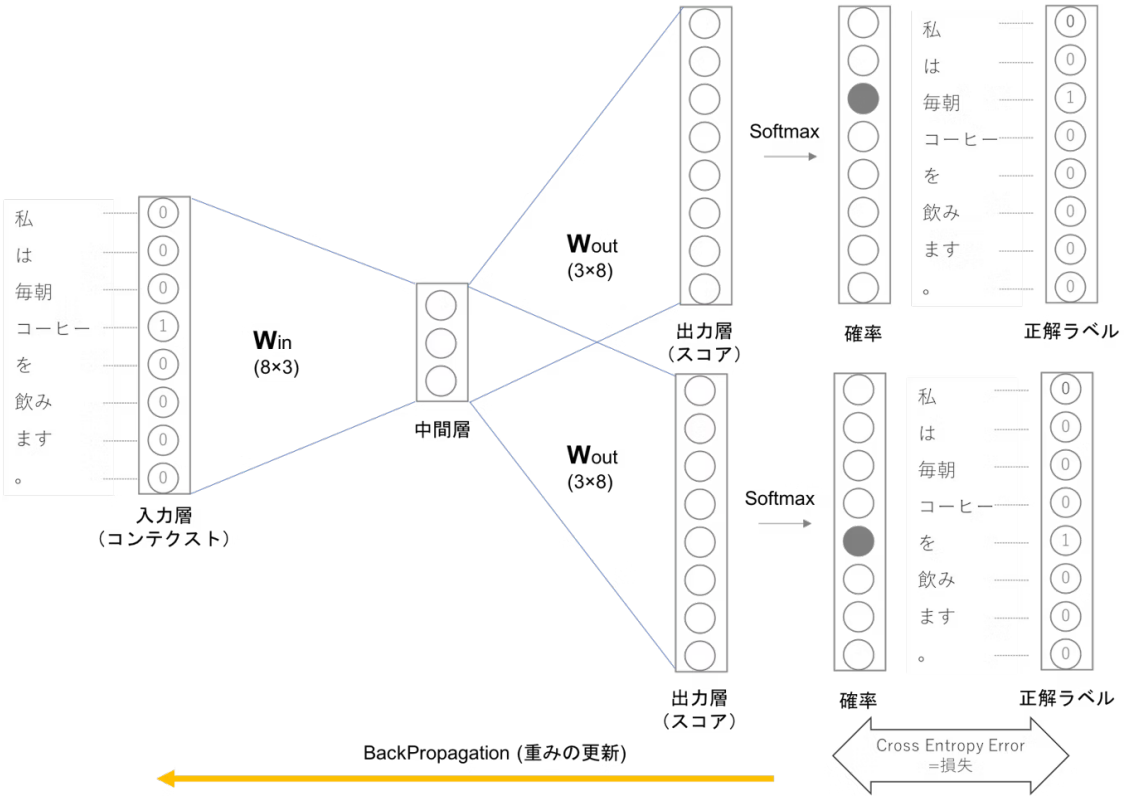


$$L = -\frac{1}{T}\sum_{t=1}^{T} logP(w_t | w_{t-1}, w_{t+1})$$

上記損失関数をできるだけ小さくするように学習がすすみ、その時の重みを単語の分散表現として獲得する。

また、Skip-gramの図

↓



また、skip-gramモデルの損失関数は（コンテクストを前後1単語とした場合）

$$L = -\frac{1}{T}\sum_{t=1}^{T}(logP(w_{t-1}|w_t) + logP(w_{t+1}|w_t))$$

となる。

6.Word2Vecの特徴

Word2Vecで学習した単語ベクトルには、次のような特徴がある。

１, 意味が近い単語同士のベクトルは近い

2,単語の関係性を計算できる

例えば、王様 - 男 + 女 = 女王
のように単語の四則演算が可能になる。

7.まとめ

・Word2Vecは 単語を数値ベクトルに変換する技術。

・CBOWとSkip-gramの2つの学習方法がある。

・単語の意味が近いもの同士のベクトルも近くなる。

・単語の関係を計算できる（例：「王様 - 男 + 女 = 女王」）

8.word2vecでの学習

上記に説明したword2vecで学習を行う。

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

```
import os
from gensim.models import word2vec
import logging

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
level=logging.INFO)
```

```
save_dir = “"#pathの指定
save_path = os.path.join(save_dir, "language_dateset3a.model") # セーブ箇所の指定

if not os.path.exists(save_dir):
    os.makedirs(save_dir)
    logging.info(f"ディレクトリ '{save_dir}' を作成しました。")

sentences = word2vec.Text8Corpus(‘tokenized_wikipedia_text.txt')
model = word2vec.Word2Vec(sentences, vector_size=200, min_count=5, window=15, sg=0, hs=0, negative=10, epochs=10, workers=8) # 後程詳しく説明
model.wv.save_word2vec_format (save_path, binary=True)
logging.info(f"モデルを '{save_path}' に保存しました。")
```

——

↑上記のプログラム（train.pyも同様にこれを使用）

また、sentences後の値はパラメータであり、

vector_size	単語ベクトルの次元数
min_count	この回数未満の単語は学習しない
window	文脈を学習する単語の範囲（前後何単語か）
Sg	モデルの種類（0=CBOW,1=skip-gram)
hs	Hierarchical Softmaxの使用有無
negative	Negative Samplingで使用する負サンプル数 （周囲の単語が出て来ない確率を学習）
epochs	エポック数（学習を何周行うか）
workers	使用するcpuコア数

ファイルはバイナリ形式で学習する(0,1の組み合わせ）

2-4④の詳細（使い方）

1 ,pythonがダウンロードしているか確認

Pythonがダウンロードしているかを確認。

※Windowsではコマンドプロンプト、macではターミナルで稼働を想定。

また、プログラムは前に○マークをつける。（コピーは○以降）

○ python —version

仮に,pythonがなかった場合、

○ winget install Python.Python.3

2,仮想環境を構築するディレクトリに移動

今回は、language_culcフォルダ内に作ることを想定するため、

C:\Users\Username\Documents\language_culc　#絶対パス

を指定し、そしてこのディレクトリに移動。

○ cd C:\Users\Username\Documents\language_culc

3,仮想環境を作成

仮想環境を作成するには、venvを使用する。

以下のコマンドで仮想環境を作成する。

○ python -m venv lang_culc

このコマンドを実行すると、language_culcという名前の仮想環境が作成される。

このディレクトリに必要なライブラリが隔離されてインストールされるようになる。

4,仮想環境を有効化

以下のコマンドで仮想環境を有効化します。

○ lang_culc\Scripts\activate

有効化されると、プロンプトが (lang_culc) という形式に変わる。これにより、仮想環境内で操作していることがわかる。

例：

(lang_culc) C:\Users\Username\Documents\lang_culc>

5,gensimをインストールする

仮想環境内で、以下のコマンドを入力してgensimをインストール。

○ pip install gensim

6pythonファイルを実行

例えば、script.pyという名前のPythonファイルがある場合、以下のコマンドで実行できる。

○ python script.py

これで、仮想環境内で実行されているPythonファイルが実行される。

7,実際に使うことのできるファイル

使うことのできるpythonファイルは以下の6つ。

add.py　　単語の足し算を実現（複数個ok)

substraction.py 単語の引き算を実現（複数個ok)

similar.py　　似ている単語を出力

similarity.py　　2つの単語の類似度を計算

vector.py　　単語のベクトル値を出力

meaningless.py最も意味の離れている単語を出力

○ python vector.py

このプログラムを実行すると、ベクトル値を出力したい単語を入力します。

○ python similar.py

このプログラムを実行すると、調べたい単語、何個の単語を昇順で出力すべきかを入力します。

○ python similarity.py

このプログラムを実行すると、調べたい2単語を入力します。

○ python add.py

このプログラムを実行すると、足し算を行いたい単語、何個の単語を昇順で出力すべきかを入力します。

○ python substraction.py

このプログラムを実行すると、引かれる単語（複数個ok)、引く単語（複数個ok)、何個の単語を昇順で出力すべきかを入力します。

○python meaningless.py

このプログラムを実行すると、調べたい単語、何個の単語を昇順で出力すべきかを入力します。

※ここでの最も意味の離れている単語というのは、対義語という意味ではなく、最も関係性のない単語となっています。

cos類似度の仕様上、対義語は目的の単語の近くにあることが多く相関関係が強い単語だと見做されるためです。

8.どのようにして実行しているのか

前述した通り、Word2Vecは単語をベクトル（数値の配列）として表現し、そのベクトルの間の距離や角度をもとに単語の類似性を計算する。

単語の類似度を測る際に、**コサイン類似度（Cosine Similarity）**が一般的に使用される。

また、コサイン類似度は以下の数式で求められる。

$$cos(a,b) = \frac{a \cdot b}{||a||\ ||b||} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$$

a=単語Aのベクトル

b=単語Bのベクトル

a・b=a.bの内積

‖A‖ =Aの ベクトルの長さ（ノルム）

‖B‖ =Bの ベクトルの長さ（ノルム）

a.b=ベクトルAとベクトルBのなす角

また、コーシーシュワルツの不等式

$$\left(\sum_{k=1}^{n} a_k^2\right)\left(\sum_{k=1}^{n} b_k^2\right) \geqq \left(\sum_{k=1}^{n} a_k b_k\right)^2$$

より、

$$-1 \leq \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \leq 1$$

である。

この時コサイン類似度は、-1〜1の範囲に正規化され，その値によって以下のように解釈が異なる。

↓

1なら「2つのベクトルの成す角度が0度 → 同じ向きのベクトル → 完全に似ている」

0なら「2つのベクトルの成す角度が90度 → 独立・直行したベクトル → 似ている/似ていないのどちらにも無関係」

-1なら「2つのベクトルの成す角度が180度 → 反対向きのベクトル → 完全に似ていない」

2つのベクトルの大きさに関わらず、2つのベクトルの向きが近いほど、類似性が高くなる。

例えば、king = [0.2, 0.8, 0.5, 0.1]

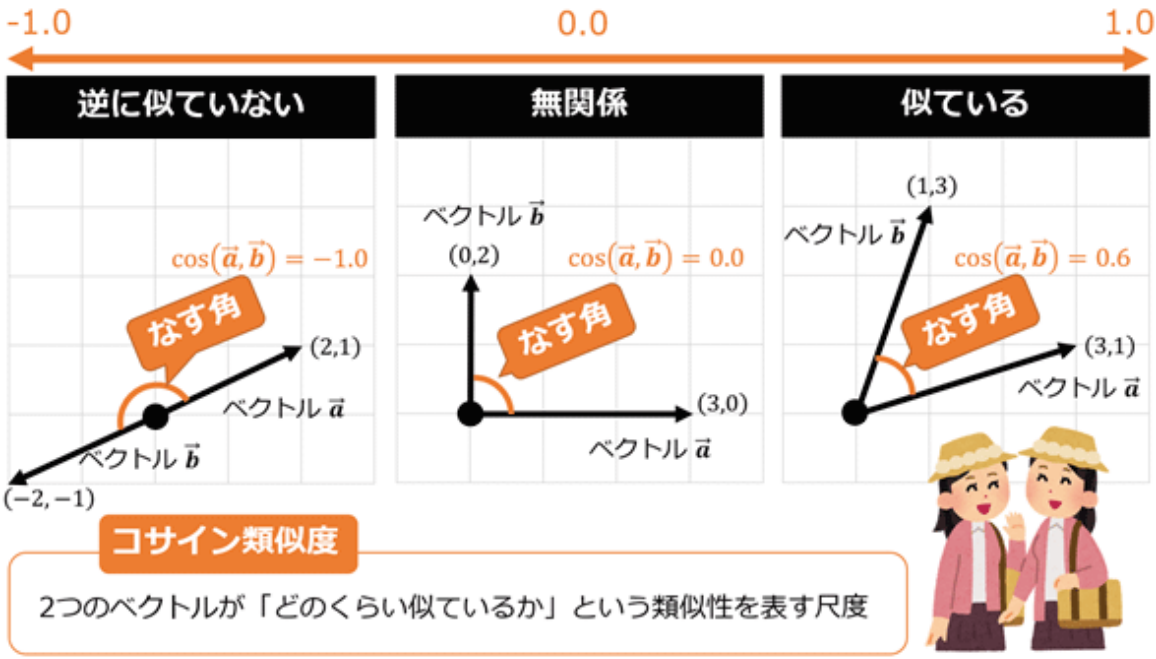
queen = [0.3, 0.7, 0.6, 0.2]
のベクトルがあった時に、
内積を計算し、

(0.2×0.3)+(0.8×0.7)+(0.5×0.6)+(0.1×0.2)=0.06+0.56+0.30+0.02=0.94
ベクトルの長さ（ノルム）を計算し、

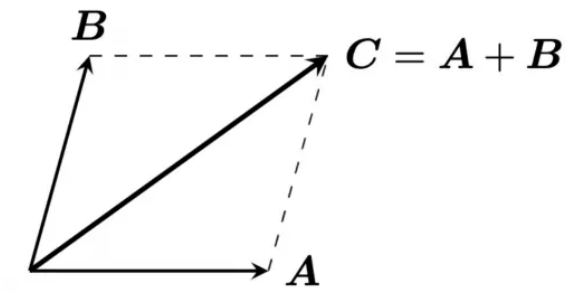
‖king‖=(0.2)2+(0.8)2+(0.5)2+(0.1)2 =0.04+0.64+0.25+0.01 =0.94 ≈0.97

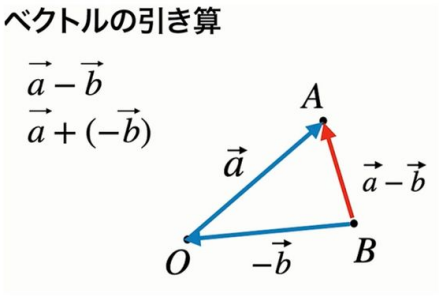
‖queen‖=(0.3)2+(0.7)2+(0.6)2+(0.2)2 =0.09+0.49+0.36+0.04 =0.98 ≈0.99
コサイン類似度を求める。

cos(θ)=0.94/(0.97×0.99) =0.94/(0.9603)≈0.978
この結果により、とても近いことがわかる。



↑上図のようになっている。





また、足し算や引き算はベクトルの合成を行っている。