

# プロジェクト第5回

単語ベクトル

n-gram

# n-gram

- テキストを連続するN個の文字またはN個の単語単位で単語を切り出す手法
- N=1：ユニグラム
- N=2：バイグラム
- N=3：トライグラム

Text：I am a NLPer

単語バイグラム：

[["I", "am"], ["am", "a"], ["a", "NLPer"]]

文字バイグラム：

[["I\_", "\_a", "am", "m\_", "\_a", "a\_", "\_N", "NL", "LP", "Pe", "er"]]

# 演習1. n-gram

与えられた文字列からn-gramを作る関数を作成せよ. また, "I am a NLPer"という文を入力として与えて単語バイグラムと文字バイグラムを確認せよ.

word2vec

# 単語の分散表現

- 単語の意味をベクトルで表したもの

例：赤 = (R, G, B) = (255, 0, 0)

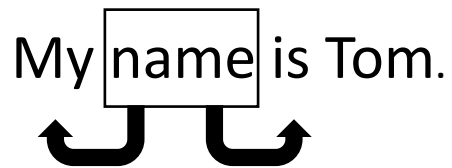
## 分布仮説

単語の意味は周囲の単語によって形成される

# ウィンドウサイズ

ウィンドウサイズ 1 の場合

My name is Tom.



このとき、Myとisがコンテキストと呼ばれる

# 共起行列

- 共起

ある文中で、ある単語とある単語が同時に使用されること

## 前ページの共起行列

	My	name	is	Tom
My	0	1	0	0
name	1	0	1	0
is	0	1	0	1
Tom	0	0	1	0



## 演習2. 共起行列の実装

共起行列を作成する関数を実装せよ。単語IDのリスト、語彙数、ウィンドウサイズを引数とする。次のページのように単語IDを作成する。???を埋める。

```
def create_co_matrix(corpus,vocab_size>window_size=1):  
    corpus_size=len(corpus)  
    co_matrix=np.zeros((vocab_size,vocab_size),  
                        dtype=np.int32)  
    ...  
    return co_matrix
```

# 単語IDの作成

```
def preprocess(text):  
    text=text.lower()  
    text=text.replace(',', ' .')  
    words=text.split(' ')  
  
    word_to_id={}  
    id_to_word={}  
    for word in words:  
        if word not in word_to_id:  
            new_id=len(word_to_id)  
            word_to_id[word]=new_id  
            id_to_word[new_id]=word  
    corpus=np.array([word_to_id[w] for w in words])  
    return corpus, word_to_id, id_to_word
```

numpyを使うと行列を表現できる

import numpy as np

インポートでエラーが出る人は  
管理者コマンドプロンプトで

pip install numpy

または

python -m pip install numpy

を入力するとインストールできる

# 演習2のソースコード

```
def create_co_matrix(corpus,vocab_size>window_size=1):
    corpus_size=len(corpus)
    co_matrix=np.zeros((vocab_size,vocab_size),dtype=np.int32)
    for idx,word_id in enumerate(corpus):
        for i in range(1>window_size+1):
            left_idx=idx-i
            right_idx=idx+i
            if ???>=0:
                left_word_id=corpus[left_idx]
                co_matrix[word_id,left_word_id]+=1
            if ???<corpus_size:
                right_word_id=corpus[right_idx]
                co_matrix[word_id,right_word_id]+=1
    return co_matrix
```

# 単語ベクトルを取り出す

```
text="My name is Tom"
```

```
corpus,word_to_id,id_to_word=preprocess(text)
```

```
vocab_size=len(word_to_id)
```

```
C=create_co_matrix(corpus,vocab_size)
```

```
my_vec=C[word_to_id["my"]] # myの単語ベクトル
```

```
print(my_vec)
```

# 共起行列の問題点

- 共起回数だと問題が生じる
- 高頻度単語「the」「a」が強い関係を持つように評価されてしまう
- この問題を解決するために、相互情報量、次元削減の手法を取り入れるようになった

# 相互情報量PMI

- $PMI(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)} = \log_2 \frac{\frac{C(x, y)}{N}}{\frac{C(x)}{N} \frac{C(y)}{N}} = \log_2 \frac{C(x, y)N}{C(x)C(y)}$

C: 共起行列, N: 単語数

- 2つの単語で共起回数が0の場合に、  
 $\log_2 0 = -\infty$  となってしまうので  
 $\rightarrow PPMI(x, y) = \max(0, PMI(x, y))$

# 演習3. PPMIの実装

共起行列をPPMIに変換する関数を実装せよ。共起行列 $C$ 、閾値 $\text{eps}=1\text{e-}8$ を引数とする。 $\text{eps}$ は相互情報量の問題点を回避するために用意する。

```
def ppmi(C, eps=1e-8):  
    M=np.zeros_like(C,dtype=np.float32)  
    N=np.sum(C)  
    S=np.sum(C,axis=0)  
    total=C.shape[0]*C.shape[1]  
    cnt=0  
    # ???を埋める  
    return M
```

# 演習3のソースコード

```
def ppmi(C, eps=1e-8):  
    M=np.zeros_like(C,dtype=np.float32)  
    N=np.sum(C)  
    S=np.sum(C,axis=0)  
    total=C.shape[0]*C.shape[1]  
    cnt=0  
    for i in range(C.shape[0]):  
        for j in range(C.shape[1]):  
            pmi=???  
            M[i,j]=max(0,pmi)  
    return M
```



# 現在のword2vec

- 管理者コマンドプロンプトで

`pip install gensim`

`pip install paramiko`

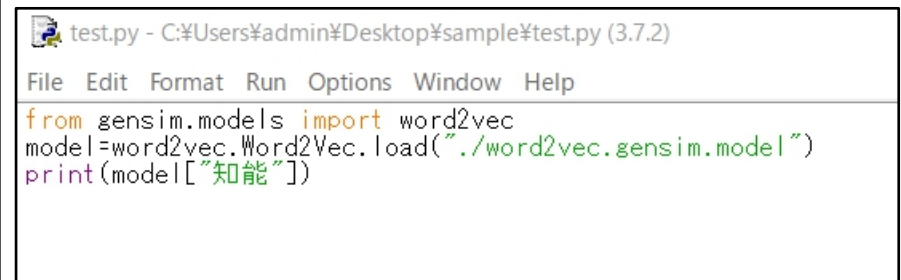
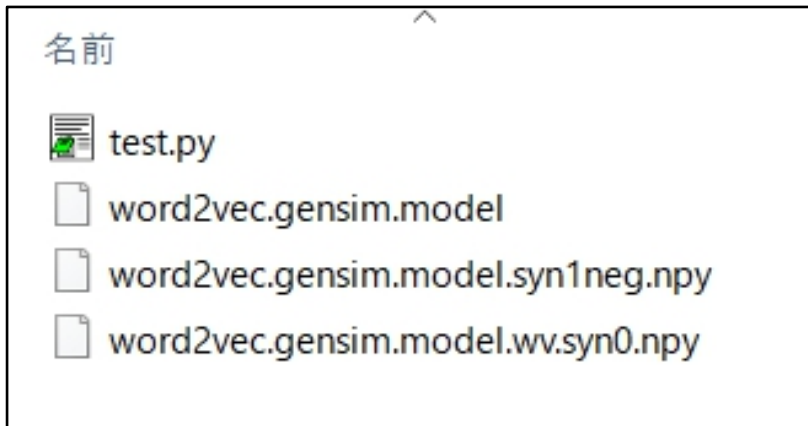
を入力する。

- <http://public.shiroyagi.s3.amazonaws.com/latest-ja-word2vec-gensim-model.zip>

からモデルをダウンロードする

# 確認する

- 前ページでダウンロードしたファイルを解凍する。解凍後のファイルの中にあるモデルを実行するプログラムと同じ場所に置く。



## 演習4. word2vec

gensimライブラリのword2vecと演習3で作成したベクトルを比較して違いを確認せよ。