理工学部生のための重要ボキャブラリー抽出

関西学院大学理工学部
情報科学科 西谷研究室
27015473
奥田健太

2020 年 2 月

目次

図目次

表目次

1　概要

関西学院大学理工学部の学生が研究室配属をされ，研究を進めていく際に英語の論文や専門書を読む機会が多い．そこで理工学部生が 1 回生の授業で開講される英語の授業で扱えるような単語帳を作れないかと考えた．単語を理工系論文サイト「PloseOne」から集め, それを元に大きな corpus を作成する。作成した corpus に id を振り分け共起行列を作成し,sin-simirality で評価をした。そこで発生する問題点である行列が大きくなったときに生じる計算時間の増大化を解消するために共起行列の次元削除を行って評価を行った。得られた結果が最適とは言えないため, 他の手法を試みた。他の手法とは一般的な単語抽出の際にもちいられる TF-IDF と呼ばれる評価式である。まずはこの TF-IDF をもちいて評価した物が有用であるかを検討し,TF-IDF よりも精巧な評価値が算出される Okapai-bm-28 を検討していたが,TF-IDF では満足な結果が得られなかった。この原因として考えられるのは TF-IDF では文書の特徴語を抽出するのが目的であるため今回の理工系の専門単語は出現回数が低く特徴語としては評価されないことが原因と考えられる。今後,TF-IDF の改良及び sin-simirality の計算速度の改善などが課題に上げられる。

2　序論

2.1　背景

関西学院大学理工学部の学生が研究室配属をされ，研究を進めていく際に英語の論文や専門書を読む機会が多い．研究を進めていく中で英語読解により時間を裂くことは効率化を測る中で無駄な時間と捉えることもできる。関西学院大学の英語の授業では二年間にわたり英語学習を進めていくが理工系論文の読解に必要な専門用語を重点的に学習する機会は多いとは言えない。EEFL の Imogen 先生に, そこで理工系論文で頻出な専門用語をリストアップし, 大学 1 回生の時から触れ合う機会があれば英語学習のモチベーションにも繋がり単語学習の向上を図れるのではないかと提案され本研究に至った。

2.2　目的

そこで理工学部生が 1 回生の授業で開講される英語の授業で扱えるような単語帳を作れないかと考えた。現在関西学院大学では TOEIC 学習のため英単語帳を購入し単語の小テストをしているが, あくまで TOEIC 学習であり専門的な単語を学場とは言えない。TOEIC 学習を進めていくのももちろんだが研究室に配属された際や将来専門職に就職した際の手助けとなるような単語帳を作成する。単語帳を作成する際実際の理工系の英語の論文から単語を集め大きな corpus を作成。corpus とは言語学において、自然言語処理の研究に用いるため、自然言語の文章を構造化し大規模に集積したもの。その中から必要な単語を集めてくる。本研究では単語を理工系の論文から抽出そして重要な単語の識別を自動化するプログラムを作成する.

3　手法

本研究では，まず，理工系の論文が掲載されているPloseOneというサイトから論文を集めた。単語の評価をしやすくするために本研究では情報科学に関する論文を集めた。論文をダウンロードしてくる際PDFファイルもしくはXMLファイルであるためtxtファイるに変換して利用した。以下のプログラムを使いXMLファイルをtxtファイルに変換したものを利用した。

Listing 1　xml2txt

```
require "rexml/document"
require "open-uri"
require 'nokogiri'
url='https://journals.plos.org/plosone/article?id=10.1371/journal.pone
    .0197607'
context = open(url).read
context.gsub!(/&(?!(?:amp|lt|gt|quot|apos);)/, '&amp;')
#doc = REXML::Document.new(context)
doc = Nokogiri::HTML.parse(context)

puts doc.title

body = false
doc.text.split("\n").each do |line|
  body = true if line.include?('Abstract')
  body = false if line.include?('References')
  puts line if body==true
end
```

表1　xml2txt.rb

集めた論文はhttps://journals.plos.org/plosone/から10個を集めた。
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0027826
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0047564
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0083553
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0091650
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0112520

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0112520
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0121855
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0160147
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0171189
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0197607
論文の語数の総数は 5 万語ほどである。
重要度の指標としては基本的な英単語には無い物で頻出度を測ることにした。集めた英単語から重要ボキャブラリーを抽出する評価方法は 3 通りを検討した。

3.0.1 nokogiri

txt ファイルに変換する際に使用した ruby ライブラリの nokogiri とはスクレイピングの際使用される物である。
特徴としては HTML や XML の構造を解析して, 特定の要素を指定しやすい形に加工できる。また,Xpath や CSS セレクタを使った要素の抽出を行うことができる。

3.1 sin-simirality による評価

自然言語処理 (Natural Language Processing:NLP) を行う．ここで行う自然言語処理 (Natural Language Processing:NLP)[4] の手法はまず，抽出した word に id を振りコーパスを作成し，次に共起行列 (図 1 は例) を作成し，特異値分解 (svd:Singular Value Decomposition) を行い次元削除をする．最後に単語間の出現確率の類似度 ($C(x)$:単語 x の出現回数,N:単語の総数,$C(x, y)$:単語 x と y の共起回数) を相互情報量 (PMI:Pointwise Mutuak Information) を元に表示し最頻単語から順に並べる.

例えば「You say goodbye, I say Hello.」という文に対しては以下の共起行列を作成する。

ある単語に着目し，周囲にどのような単語が現れるかをカウントし共起行列を作成する．1 文に 2 回出現する単語は 2 回表には入れない.

$$A = USV^T \tag{1}$$

共起行列 A を 3 つの行列に特異値分解をする．特異値分解を行う理由としては, 単語からなる共起行列が正方行列であるとは限らないためである。ここで U と V は直交行列で S は対角行列．直交行列 U は何らかの空間の軸 (基底) を形成するので，ここで U を「単語空間」として扱うことができる．S は対角行列で対応する「軸の重要度」である「特異値」が大きいものの順に並んでいる．次元削除とはここから重要でない軸を削除し重要なデータだけを残すことができる．次元削除を行うことで行列の大きさを小さくし計算を効率化させることができる.

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1		1	0	0
I	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

図 1　共起行列の例.

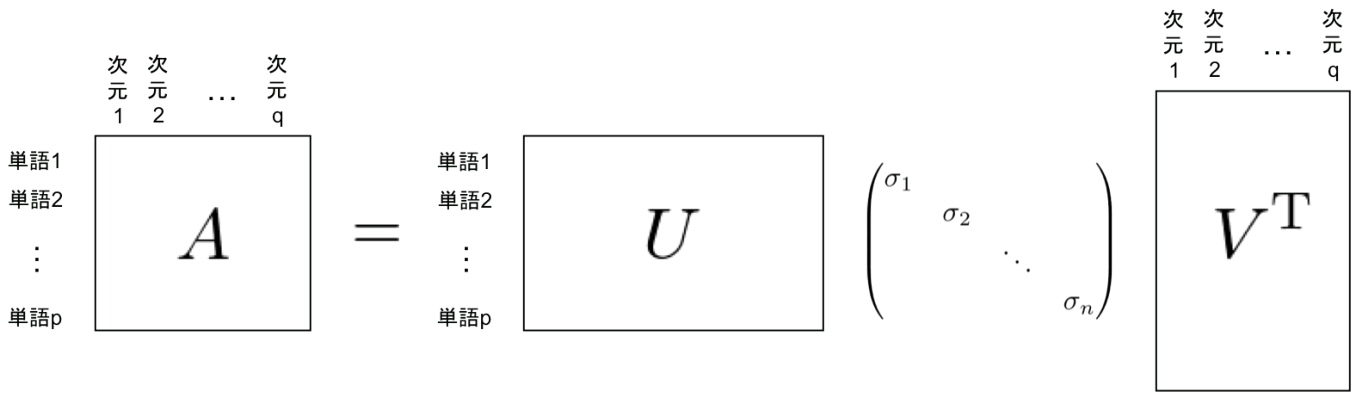

図 2　次元削除のイメージ.

$P(x,y)$ は単語 x,y が共起する確率,$P(x)$ は単語 x が出現する確率として以下のように定義される

$$PMI(x,y) = \log_2 \frac{P(x,y)}{P(x)(y)} = \log_2 \frac{\frac{C(x,y)}{N}}{\frac{C(x)}{N}\frac{C(y)}{N}} = \log_2 \frac{C(x,y)N}{C(x)C(y)} \tag{2}$$

2 つの単語で共起する回数が 0 の場合，$\log_2 0 = -\infty$ となってしまうので正の相互情報量 (PPMI:Positive Pointwise Mutual Information) を用いる．コードは ruby で書き, 実装したプログラムは以下の通りである。

まず,preprocess で word に id を割り振り,corpus を作成する。

Listing 2　preprocess

```
  def preprocess(text)
  text.gsub!('.', ' .')
  text.gsub!(',', ' ,')
  words = text.downcase.split(/\s+/)
  word_to_id = {}
  id_to_word = {}
  words.each do |word|
    unless word_to_id.include?(word)
      new_id = word_to_id.size
      word_to_id[word] = new_id
      id_to_word[new_id] = word
    end
  end
  corpus = []
  words.each do |word|
    corpus << word_to_id[word]
  end
  return corpus, id_to_word, word_to_id
end
```

表 2　preprocess.rb

次に create_co_matrix を使い共起行列を作成。

Listing 3　create_co_matrix

```
  def create_co_matrix(corpus, vocab_size, window_size=1)
  corpus_size = corpus.size
  co_matrix = Array.new(vocab_size){Array.new(vocab_size,0)}
  corpus.each_with_index do |word_id, idx|
    word_id
    (1..window_size).each do |i|
      left_idx = idx - i
      right_idx = idx + i
      if left_idx >=0
        left_word_id = corpus[left_idx]
        co_matrix[word_id][left_word_id] += 1
      end
      if right_idx < corpus_size
        right_word_id = corpus[right_idx]
```

```
15          co_matrix[word_id][right_word_id] += 1
16        end
17      end
18    end
19    co_matrix
20  end
```

ppmi で positive pairwise mutual info を作成

Listing 4　ppmi

```
1   def ppmi(co_matrix, eps = 1e-8)
2   i_n = j_n = co_matrix[0].size
3   total = i_n * j_n
4
5   nn = 0.0
6   i_n.times do |i|
7     j_n.times do |j|
8       nn += co_matrix[i][j]
9     end
10  end
11
12  ss = Array.new(i_n, 0.0)
13  i_n.times do |i|
14    j_n.times do |j|
15      ss[i] += co_matrix[i][j]
16    end
17  end
18
19  mm = Array.new(i_n){Array.new(j_n,0.0)}
20  i_n.times do |i|
21    j_n.times do |j|
22      pmi = Math.log2(co_matrix[i][j]*nn/(ss[j]*ss[i])+eps) # 0.0 : pmi
23      mm[i][j] = (pmi < 0.0 )? 0.0 : pmi
24    end
25  end
26  mm
27 end
```

そして most_similarity で cos_similarrity を表示

表 3　ppmi.rb

Listing 5　most_similarity

```
1   def most_similary(query, word_to_id, id_to_word, ww, u, top=5)
2   unless word_to_id.include?(query)
3     printf('%s is not found\n', query)
4     exit
5   end
6
7   printf("\n * Similar usage words to [%s]\n", query)
8   query_id = word_to_id[query]
9   query_vec= ww[query_id]
10
11  printf("\n ** cos similarity\n")
12  vocab_size = id_to_word.size
13  similarity = Array.new(vocab_size, 0.0)
14  vocab_size.times do |i|
15    similarity[i] = cos_similarity(ww[i], query_vec)
16  end
17  rank = similarity.map.with_index.sort.map(&:last).reverse
18  rank[0..top-1].each do |count|
19    printf("%20s: %10.5f\n", id_to_word[count], similarity[count])
20  end
21
22  printf("\n ** ppmi+svd coordination \n")
23  u0 = [u[query_id,0],u[query_id,1]]
24  vocab_size.times do |i|
25    similarity[i] = distance(u0, [u[i,0],u[i,1]])
26  end
27  rank = similarity.map.with_index.sort.map(&:last)
28  rank[0..top-1].each do |count|
29    printf("%20s: %10.5f\n", id_to_word[count], similarity[count])
30  end
31 end
```

表 4　sin-simirality.rb

closest_word で co_vector から最頻単語を表示

Listing 6　closest_word

```
def closest_word(query, co_matrix, word_to_id, id_to_word, top=5)
  printf(" ** closest word to [%s]\n", query)
  co_vec = co_matrix[word_to_id[query]]
  rank = co_vec.map.with_index.sort.map(&:last).reverse
  rank[0..top-1].each do |count|
    printf("%20s: %7d\n", id_to_word[count], co_vec[count])
  end
end
```

表 5　closest-word.rb

3.2　TF-IDF

TF-IDF を用いた単語の重要度を算出する手法を用いる.TF-IDF とは文書内に出現する単語について, TF(Term frequency) とは単語中に特定の単語がどれくらい登場するかを示す指標である。IDF(Inverse Document Frequency) は特定の単語がどれくらい様々な文書で登場するかを示す指標である。

$$tf(t_i, d_j) = \frac{f(t_i, d_j)}{\sum_{t_k \in d_j} f(t_k, d_j)} \tag{3}$$

ある文書 d_j に出現する単語 t_j について出現回数を f とする.

$$idf(t_i) = \log \frac{N}{df(t_i) + 1} \tag{4}$$

df を単語 t_i が出現する文書数とする.
IDF 値は文書集合の中にある単語が含まれる文書の割合の逆数を表す. 単語が他の文書にも多く出現しているほど小さくなり, 出現していなければ大きくなる.

$$tfidf(t_i, d_j) = tf(t_i, d_j) \cdot idf(t_i) \tag{5}$$

手法 1 の例 (図 1:共起行列の例) にある文書 A「You say goodbye and I say hello.」ある文書 B「Hello, I am a student」について TF-IDF を考えてみると,

tf(You,A)=$\frac{1}{7}$, tf(say,A)=$\frac{2}{7}$,tf(goodbye,A)=$\frac{1}{7}$,tf(and,A)=$\frac{1}{7}$

tf(I,A)=$\frac{1}{7}$,tf(hello,A)=$\frac{1}{7}$,tf(hello,B)=$\frac{1}{5}$,tf(I,B)=$\frac{1}{5}$,

tf(am,B)=$\frac{1}{5}$,tf(a,B)=$\frac{1}{5}$,tf(student,B)=$\frac{1}{5}$,

	You	say	goodbye	and	I	hello	Student
A	0.14	0.3	0.14	0.14	0.14	0.14	0
B	0	0	0	0	0.2	0.2	0.2

図 3　tf 例

となり idf 値を計算すると,

idf(You)=$\log_2 \frac{2}{1}$=1.0, idf(say)=$\log_2 \frac{2}{1}$=1.0, idf(goodbye)=$\log_2 \frac{1}{1}$=0, idf(and)=$\log_2 \frac{1}{1}$=0,

idf(I)=$\log_2 \frac{2}{2}$=0, idf(hello)=$\log_2 \frac{2}{2}$=0 idf(am)=$\log_2 \frac{1}{1}$=0, idf(a)=$\log_2 \frac{1}{1}$=0 idf(student)=$\log_2 \frac{1}{1}$=0

tfid 値を計算したもが以下の (図 5)

	You	say	goodbye	and	I	hello	Student
A	0.14	0.3	0.14	0.14	0	0	0
B	0		0	0	0	0	0.2

図 4　tfidf 例

となり 2 文書に出現している単語は 0 となり文書の特徴を表していないこととなる.
この手法を用いて重要単語を評価したいが, プログラムの作成及び, 単語の出現文書数の値を利用するため文書の単語数により差が生じないかを検討していく必要がある.
コードは python で書き, プログラムは以下の通りである。

Listing 7　TF-IDF

```
ファイルの読み込み
#
with open('corpus.txt', "rU") as f:
    corpus = map(lambda x:x.split("\t"), f.read().strip().split("\n"))
with open('corpus.txt', "rU") as f:
    corpus = [v.rstrip() for v in f.readlines()]

import numpy as np
import pandas as pd
import re
from collections import defaultdict不要な記号の削除

#
documents = [re.sub('[.|?]', '', i.lower()) for i in corpus]スペースごとに区切って格納

#
documents = [doc.split(' ') for doc in documents]基本単語の削除
#
print(documents)

stop_words = []
#stop_words = ["is", 'this']
f = open("basic_voc.txt", "r")
lines = f.readlines()
for line in lines:
  for word in line.split(','):
    stop_words.append(word)

documents_deleted = []
for doc in documents:
    documents_deleted_temp = []
    for word in doc:
        if word not in stop_words:
            documents_deleted_temp.append(word)
    documents_deleted.append(documents_deleted_temp)


documents = documents_deleted
print(documents_deleted)全文書中の単語の種類を取得

```

```
#
vocab = defaultdict()
vocab.default_factory = vocab.__len__
i = 0
for doc in documents:
    feature_counter = {}
    for term in doc:
        if term not in vocab.keys():
            vocab[term] = i
            i+=1


#の出現カウントベクトル term
x = np.zeros(shape=(len(corpus), len(vocab)), dtype=int)
for idx, doc in enumerate(documents):
    for word in doc:
        if word in vocab.keys():
            x[idx, vocab[word]] += 1

#tf
N = len(x)
tf = np.array([x[i, :] / np.sum(x, axis=1)[i] for i in range(N)])

#idf
df = np.count_nonzero(x, axis=0)
idf = np.log((1 + N) / (1 + df)) + 1

tfidf = tf*idf
tfidf = pd.DataFrame(tfidf, columns=vocab.keys())
print(tfidf)
```

表 6　TF-IDF.py

Qiita(https://qiita.com/katryo/items/f86971afcb65ce1e7d40)

3.2.1　pandas

ここで用いた pandas とは Python において、データ解析を支援する機能を提供するライブラリである。特に、数表および時系列データを操作するためのデータ構造と演算を提供す

る物である。
pandas の主な特徴は以下の通りである。
・データ操作のための高速で効率的なデータフレーム (DataFrame) オブジェクト
・メモリ内のデータ構造とその他のフォーマットのデータ間で相互に読み書きするためのツール群。フォーマット例: CSV、テキストファイル、Excel、SQL データベース、HDF5 フォーマットなど
・かしこいデータのアライメントおよび統合された欠損値処理
・データセットの柔軟な変形およびピボット
・ラベルに基づいたスライス、fancy インデクシング、巨大なデータセットのサブセット取得
・データセットに対する split-apply-combine 操作を可能にするエンジンが提供する powerful group を使ったデータの集計および変換
・高性能なデータセットのマージと結合
・時系列データ: 日付範囲生成、周波数変換、移動窓を用いた統計値や線形回帰、シフトと遅延、などパフォーマンスのための高度な最適化。重要なコードは Cython または C 言語で実装されている。
引用:Qiita「データ分析で頻出の Pandas 基本操作」(https://qiita.com/ysdyt/items/9ccca82fc5b504e7913

3.3 Okapai bm25

TF-IDF 値と同様,Okapai BM25 では単語ごとの重要度を算出する手法である。Okapai BM25 も TF-IDF 同様に算出した重要度は文書ごと, 単語ごとに存在する。

$$score(t_k, d_j) = idf(t_k) \frac{tf(t_k, d_j)(k_1 + 1)}{tf(t_k, d_j) + k_1(1 - b + b\dfrac{dl(d_j)}{avgdl})} \tag{6}$$

$score(t_k, d_j)$ の値が大きくなるほど重要度が高いことを示す。また, 分子は単語おの出現頻度から算出した TF-IDF 値に関する計算部分であり, 算出式の分母は文書の単語数に関する計算部分である。
重要度の指標としてこの値は
(1) 文書における単語の出現頻度 (TF 値) が高い
(2) 文書集合での出現頻度 (DF 値) が低い (IDF 値が高い)
(3) 単語数 (DL 値) が少ない文書に出現している
これらを満たすような単語の重要度が高くなる。
式中のパラメータに関して,k_1 は単語の出現頻度から計算した重要度 (TF-IDF 値の指す) の影響の大きさを調整するパラメータのことで,$k_1 = 1.2$ もしくは 2.0 とする。
b は主に文書の単語数による影響の大きさを調整するパラメータであり,0.0 1.0 の間で設定

し, 今回は b=0.75 とする。$b = 0.0$ とした場合は文書の単語数による影響をなくした結果を得ることができる。
先ほどの list7 に示した TF-IDF のコードに総単語数を増やし,Okapai Bm 25 のスコアを算出した。

4　結果

4.1　sin-simiralyty による結果と考察

Listing 8　出力結果

```
 * commands

# > ruby xml2txt.rb 'https://journals.plos.org/plosone/article?id
    =10.1371/jornal.pone.0027826'
   > tmp.txt
   > ruby text_sim_proc.rb
 num of words: 7097
 * frequntly appeared word list
                not: 200: -0.05 - +0.01
               data: 181: -0.05 - +0.01
              their: 175: -0.06 - +0.03
        information: 169: -0.05 - +0.02
        performance: 153: -0.03 - +0.00
           students: 153: -0.03 - -0.00
          citations: 152: -0.03 - -0.01
            between: 151: -0.05 - +0.03
                all: 151: -0.05 - +0.01
              image: 149: -0.00 - -0.00
             school: 147: -0.02 - +0.01
               have: 142: -0.05 - +0.02
              study: 129: -0.04 - +0.01
              using: 128: -0.06 - +0.03
                can: 127: -0.05 - +0.02
            science: 127: -0.03 - +0.01
               more: 126: -0.05 - +0.01
              these: 120: -0.05 - +0.01
               than: 116: -0.04 - +0.02
           articles: 115: -0.04 - -0.00
                but: 114: -0.04 - -0.00
            results: 113: -0.03 - -0.01
type query words or return for quit:
```

この結果をみると数値の高い順に「not」や「their」は基本単語と呼べるので今回抽出する単語には含まれない。しかし,「data」「citiations」などは理工系の論文に頻出な単語と言える

表 7　出力結果.rb

だろう。また今回の実験は 5 万語の corpus を用いて行い 36 秒の実行時間を要した。今後の展望としてはこのプログラムの実行時間の短縮が上げられる。特に途中の大きな corpus を重要度が高いものだけを残すために特異値分解を行っているが特異値分解の性能の向上も検討できる。

4.2　TF-IDF と Okapai bm25 による結果と考察

Listing 9　出力結果

```
1       [['a', 'program', 'for', ... 'dorothy', 'munjalu', 'hospital)']]
2 0 0.000015 0.000601 0.010086 ... 0.000015 0.000015 0.000015
3 1 0.000015 0.000601 0.010086 ... 0.000015 0.000015 0.000015
4
5 [2 rows x 8320 columns]経過時間：
6 171.9379439353943
```

表 8　出力結果

一般的に単語抽出の代表的な物が TF-IDF や Okapai bm 25 であるが今回の専門的な単語を抽出することに関してはより良いデータが得られなかった。原因の 1 つとして基本単語の出現回数が多く, それらが特徴語として捉えらえ評価されたのが原因だと考えられる。そこで, 今回の研究ではこの評価方法は有用でないと考えた。Okapai bm25 に関しても TF-IDF の計算精度をあげる手法なので今回は同じく有用でないと考えた。

5　総括

本研究では, 単語抽出の中でも理工系の専門単語という出現回数の少ない単語の抽出を目標として評価方法の検討を行った。いかに研究内容を簡潔に示す。

1. 論文から単語テキストファイルを作成ができる。
2. 共起行列を作成。
3. 次元が大きくなるため次元削除を行い次元を小さくする。
4.sin-simirality による評価
5.TF-IDF による評価

今後の展望としては, 基本単語の出現回数などを加味して評価前の corpus の操作を行うプログラムの作成。それにより TF-IDF による評価値の改善。実験値で比較的有用な結果がでた sin-simirality の計算速度の改善及び計算精度の向上を行う。

6　謝辞

本研究を行うにあたり，終始多大なる御指導，御鞭撻をいただいた西谷滋人教授に対し，深く御礼申し上げます．また，本研究の進行に伴い，様々な助力，知識の供給を頂きました西谷研究室の同輩，先輩方に心から感謝の意を示します．本当にありがとうございました．

参考文献

[1]「Ruby によるクローラー開発技法」佐々木拓朗，るびきち，(SB クリエイティブ, 2014).

[2]「Python クローリング＆スクレイピング」加藤耕太，(技術評論社, 2016).

[3]「データを集める技術」佐々木拓朗, (SB クリエイティブ, 2016).

[4]「ゼロから作る Deep Learning 2 自然言語処理編」斎藤康毅, (オライリージャパン, 2018).

[5]「(技術解説) 単語の重要度を測る?TF-IDF と Okapi BM25 の計算方法とは」(https://mieruca-ai.com/ai/tf-idf_okapi-bm25/#toc_2)

[6]「tf-idf の実装」Qiita(https://qiita.com/tsugar/items/0391c9a45842f9d9ae69)

[7]「Okapi BM25 をスパース行列のまま計算する Python ライブラリを作った」(https://kujira16.hateblo.jp/entry/2016/01/20/235500)

[8]「英語論文から単語を抽出&登場回数順にソートし、さらに意味も載った単語帳まで作ってみた。」(https://qiita.com/mkunu/items/9b59e77de964a109e46b)

[9]「pandas」http://pandas.pydata.org/pandas-docs/stable/

7　参考

使用した corpus を単語の出現回数によりリストアップしたもの

Listing 10　ソート結果

```
1 ht236
2 sam130
3 health122
4 was106
5 were96
6 information92
7 medical82
8 hospitals77
9 or76
10 study72
11 whi72
12 use68
13 hormone68
14 months67
15 physicians67
16 data65
17 therapy64
18 their61
19 severe60
20 management60
21 risks60
22 pmid:59
23 mortality58
24 aged58
25 women58
26 hospital57
27 one56
28 patients54
29 1371/journal54
30 we52
31 among51
32 pone51
33 children50'
```

```
womens49
may48
media48
benefits47
had46
all46
not46
acute45
more45
malnutrition43
plos43
https://doi43
org/43
care42
```

ソートする際に使用したコード

```
import os
import re
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
from io import StringIO

pdf_folder_path = os.getcwd() + '/' + 'pdf'
# 現在のフォルダのパスを取得
text_folder_path = os.getcwd() + '/' + 'text'
# の表記が仕様。の場合は、pathmacwindowsをに修正する。'/''\'

os.makedirs(text_folder_path, exist_ok=True)
pdf_file_name = os.listdir(pdf_folder_path)

# が（末尾が namepdf.）の場合は、それ以外はを返す関数 pdfTrueFalse
def pdf_checker(name):
    pdf_regex = re.compile(r'.+\.pdf')
    if pdf_regex.search(str(name)):
        return True
    else:
        return False
        def convert_pdf_to_txt(name, txtname, buf=True):
```

```
25      rsrcmgr = PDFResourceManager()
26      if buf:
27          outfp = StringIO()
28      else:
29          outfp = file(txtname, 'w')
30      codec = 'utf-8'
31      laparams = LAParams()
32      laparams.detect_vertical = True
33      device = TextConverter(rsrcmgr, outfp, codec=codec, laparams=laparams
            )
34
35      fp = open(pdf_folder_path + '/' + name, 'rb') # を開く。name
36      interpreter = PDFPageInterpreter(rsrcmgr, device)
37      for page in PDFPage.get_pages(fp):
38          interpreter.process_page(page)
39      fp.close()
40      device.close()
41      if buf:
42          text = outfp.getvalue()
43          make_new_text_file = open(text_folder_path + '/' + name + '.txt
                ', 'w') # name.を開く。
              txt
44          make_new_text_file.write(text)
45          make_new_text_file.close()
46      outfp.close()
47  # 各をそれぞれに変換。pdftxt
48  for name in pdf_file_name:
49      if pdf_checker(name):
50          convert_pdf_to_txt(name, name + '.txt')
51          # を使い、（末尾が pdf_checkerTrue.の場合）は変換に進む。pdf
52      else:
53          pass
54      # ファイルでない場合には pdfpass
55      txt_file_name = os.listdir(text_folder_path)
56  mer = open('data.txt', 'w', encoding='UTF-8', newline='')
57  # data.を新規作成（初期化）txt
58  mer.close()
59
60  # が（末尾が nametxt.）の場合は、それ以外はを返す関数 txtTrueFalse
61  def txt_checker(name):
62      txt_regex = re.compile(r'.+\.txt')
```

```
63      if txt_regex.search(str(name)):
64          return True
65      else:
66          return False
67
68  # をマージ（統合）する関数 txt
69  def txt_merge(name):
70      f = open(name, 'r', encoding='UTF-8', newline='')
71      mer = open('data.txt', 'a', encoding='UTF-8', newline='')
72      mer.write(f.read())
73      f.close()
74      mer.close()
75  # 各を１つにまとめる。txt
76  for name in txt_file_name:
77      if txt_checker(name):
78          txt_merge(text_folder_path + '/' + name) # を使い、（末尾が
                txt_checkerTrue.の場合）は変換に進む。txt
79      else:
80          pass # ファイルでない場合には txtpass
81      from collections import Counter
82  import csv
83
84  f = open('data.txt', 'r', encoding='UTF-8') # 作った data.を読み込む。
        txt
85  target_text = f.read()
86  f.close()
87
88  # 各単語の登場回数を調べる。
89  words = re.split(r'\s|\,|\.|\(|\)', target_text.lower())
90  counter = Counter(words)
91  from DictionaryServices import DCSGetTermRangeInString,
        DCSCopyTextDefinition
92
93  # 既に知っている単語のリストを作成
94  alreadyknown = ["the","of","and","in","to","is","for","that","by","
        this","as","are","be","on","with","from","an","which"]
95
96  f = open('data.csv', 'w', encoding='UTF-8', newline='') # data.を新規作
        成（初期化）csv
97  csvwriter = csv.writer(f)
98  label = ['wordlist', 'count', 'definition']
99  csvwriter.writerow(label)
```

```
100 f.close()
101 for word, count in counter.most_common():
102     csvlist = [] # を初期化 csvlist
103     if len(word) < 2 or word in alreadyknown or count < 2:
104         # 一文字の単語、既に知っている単語（リストにある)、登場回数が一回しかないなら飛ばす。alreadyknown
105         pass
106  else:
107         csvlist.append(word) # 列目は単語名 1
108         csvlist.append(count) # 列目は単語の登場回数 2
109                 f = open('data.csv', 'a', encoding='UTF-8', newline='')
110         csvwriter = csv.writer(f)
111         csvwriter.writerow(csvlist) # を末尾へ追加 csvlist
112         f.close()
```

(引用:https://qiita.com/mkunu/items/9b59e77de964a109e46b)