PythonでTF-IDF計算

専門演習(B)：Pythonでデータ分析入門

TF-IDFの概念

- 文書群内の特定の文書における特定の単語の重要度
- 仮定：以下のような単語の重要度は高い
  - その文書における出現頻度が高い
  - 文書群の他の文書にはあまり出現しない

TF-IDFの定義

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

$\text{tf}(t, d)$: 単語 t の文書 d における出現頻度

$$\text{tf}(t, d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

$$f_{t,d} = \frac{n_{t,d}}{\sum_{t'} n_{t',d}}$$

$\text{idf}(t, D)$: 単語 t の文書群 D における出現頻度

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

TF-IDFの例

	あたり	あなた	いた事	ヴァイオリン	おまえ
人間失格	0	31	0	0	0
走れメロス	0	0	0	0	20
吾輩は猫である	0	85	0	73	0
こころ	0	187	0	0	
それから	0	0	0	0	
鼻	0	0	0	0	
蜘蛛の糸	3	0	0	0	
羅生門	0	0	3	0	

← 文書ごとの単語の出現頻度（回数）

TF-IDF →

	あたり	あなた	いた事	ヴァイオリン	おまえ
人間失格	0.00	0.26	0.00	0.00	0.00
走れメロス	0.00	0.00	0.00	0.00	0.89
吾輩は猫である	0.00	0.24	0.00	0.28	0.00
こころ	0.00	0.30	0.00	0.00	0.00
それから	0.00	0.00	0.00	0.00	0.00
鼻	0.00	0.00	0.00	0.00	0.00
蜘蛛の糸	0.34	0.00	0.00	0.00	0.00
羅生門	0.00	0.00	0.25	0.00	0.00

✓ 回数の多い単語が重要というわけでもない

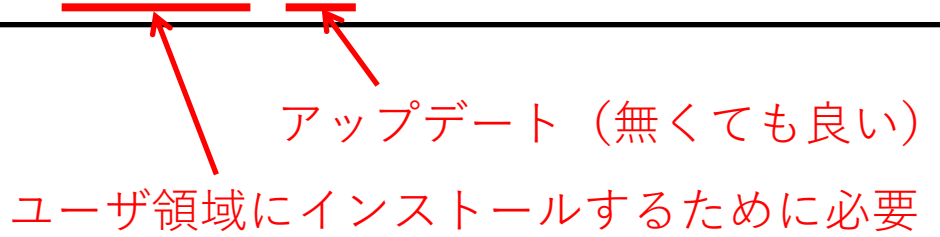
準備

必要なライブラリのインストール

- TF-IDFの計算に必要なライブラリ
  - scikit-learn
    - 機械学習とデータマイニング用のライブラリ

  [以下はscikit-learnで必要]
  - numpy
    - 数値演算用のライブラリ（特に行列演算）
  - scipy
    - 科学技術計算用のライブラリ
- インストール手順（青字が入力）

```
$ pip install ライブラリ名 --user -U
```



演習用データの作成(1)

- テキスト文書から単語リストを作成する

[手順]

1. プログラムとテキスト文書ファイル（8つ）をダウンロード
2. ターミナルで以下のように実行してみる

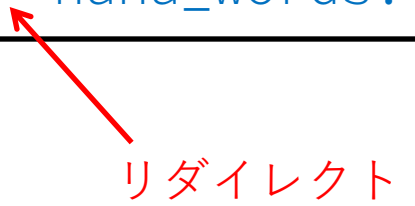
```
$ python make_word_list.py hana.txt
芥川龍之介 腸詰め まん中 何より 鼻の先 自尊心 仕合せ 自尊心 自尊心 いろいろ ため息 観音経 鼻の先 心やり 舎利弗 わざわざ ほとんど 長楽寺 いつも 気の毒 禿げ頭 じゃが 独り言 不愉快 八の字 あたりまえ ほとんど まだら 鼻の先 あくる日 法華経 それまで 禿げ頭 さかえ こっち 傍観者 二言目 しまい 手つき うすい ほとんど あけ方 芥川龍之介 ちくま 筑摩書房 1986 1997 芥川龍之介 筑摩書房 1971 1971 新思潮 1916 平山誠 もりみつ じゅんじ 1997 2011 ファイル ファイル インターネット 図書館 http www aozora ボランティア 皆さん
```

テキスト文書に含まれる単語のリスト

演習用データの作成(2)

3. 単語リストを画面で確認したら以下を実行

```
$ python make_word_list.py hana.txt > hana_words.txt
```



4. これを残りのテキストファイル（7つ）について行う

```
$ python make_word_list.py kokoro.txt > kokoro_words.txt
$ python make_word_list.py kumo.txt > kumo_words.txt
$ python make_word_list.py merosu.txt > merosu_words.txt
$ python make_word_list.py ningen.txt > ningen_words.txt
$ python make_word_list.py rasho.txt > rasho_words.txt
$ python make_word_list.py sorekara.txt > sorekara_words.txt
$ python make_word_list.py waganeko.txt > waganeko_words.txt
```

リダイレクト

- シェルの機能：入出力先を変更する

記号	機能	使用例
>	画面表示をファイルに上書き保存する	`$ ls > file_list.txt`
>>	画面表示をファイルに追加する	`$ date >> file_list.txt`
<	キーボード入力の代わりにファイルから入力する	

対話モードでTF-IDF

Pythonの起動

- 何はともあれまずはPythonの起動

```
$ python
Python 2.7.11 (default, Dec 15 2015, 19:21:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Pythonの対話用プロンプト

必要なライブラリの読み込み

- TF-IDF計算用ライブラリ（scikit-learnの一部）を読み込み

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
```

単語リストの読み込み

- 単語リストを読み込む

```
>>> f = open('hana_words.txt', 'r')
>>> hana = f.read()
>>> f.close()
```

- これを全ての単語リストについて行う

```
>>> kokoro = open('kokoro_words.txt', 'r').read()
>>> kumo   = open('kumo_words.txt',   'r').read()
>>> merosu = open('merosu_words.txt', 'r').read()
>>> ningen = open('ningen_words.txt', 'r').read()
>>> rasho  = open('rasho_words.txt',  'r').read()
>>> sorekara = open('sorekara_words.txt', 'r').read()
>>> waganeko = open('waganeko_words.txt', 'r').read()
```

✓ 横着して1行でファイルのオープンと読み込みをしている
✓ あまり良くないがファイルのクローズは省略

単語リストの確認

- 単語リストの確認

```
>>> print("%s" % merosu)
>>> 走れメロス 太宰治 メロス メロス メロス きょう メロス 山越え シラクス メロス 結婚式 メロス 御馳走 メロス セリヌンティウス シラクス つもり 楽しみ メロス 市全体 のんき メロス 若い衆 賑やか 若い衆 こんど メロス からだ あたり たくさん はじめ 御自身 お疑い ひとり 御命令 十字架 きょう メロス メロス 買い物 メロス メロス つもり ディオニス メロス おまえ おまえ メロス 心構え おまえ かたまり こんど メロス おまえ 命乞い メロス 足もと つもり 結婚式 メロス セリヌンティウス 生意気 嘘つき 身代り 身代り 正直者･･･
```

ドキュメントセットの作成

- 文書ごとの単語リストのリストを作成する

```
>>> docs = [hana, kokoro, kumo, merosu, ningen, rasho, sorekara,
waganeko]
```

TF-IDFの準備

- TF-IDF計算プログラムの用意

```
>>> vectorizer = TfidfVectorizer(use_idf = True,
                                 stop_words = ['http', 'www', 'aozora'])
```

［TF-IDFのオプション(一部)］

- use_idf
  - IDFを利用するかどうか（True|False）
- stop_words
  - TF-IDFの計算に含めない単語のリスト

TF-IDF計算の実行

- TF-IDF計算の実行

```
>>> tfidfs = vectorizer.fit_transform(docs)
```

単語リストの確認

- 単語リストの取得

```
>>> terms = vectorizer.get_feature_names()
```

- 単語リストの表示

```
>>> print('%s' % ' '.join(terms))
```

termsの中身を' '（半角スペース）で結合する

TF-IDFの確認

- TF-IDFの表示

0番目の文書(hana.txt)のTF-IDF

```
>>> tfidfs.toarray()[0]
>>> array([ 0.     ,  0.     ,  0.12455,  0.18015,  0.     ,
         0.     ,  0.09007,  0.     ,  0.     ,  0.18015,
         0.     ,  0.     ,  0.     ,  0.     ,  0.09007,
...
```

単語リストの0番目の単語のTF-IDF

主成分分析してみる

主成分分析の準備

- ライブラリの読み込み

```
>>> from sklearn.decomposition import PCA
```

- 主成分分析器の用意

```
>>> pca = PCA(n_components = 0.9, whiten = False)
```

主成分分析してみる

- 主成分分析の実行

```
>>> pca.fit(tfidfs.toarray())
```

- 主成分の数の確認

```
>>> pca.n_components_
>>> 6
```

因子負荷量の確認

- 因子負荷量の多い/少ない単語を主成分毎に表示してみる
  → 主成分の意味を推定する

```
>>> for i in range(pca.n_components_):
...     tc = zip(pca.components_[i], terms)
...     tc.sort()
...     tc.reverse()
...     print("[PCA%d]" % (i+1))
...     for v,t in tc[:10]:
...             print("%f,%s" % (v,t))
```

こいつをなくすと因子負荷量の少ない順に表示

因子負荷量の例

多い順

PCA1
0.194406,メロス
0.134848,西洋人
0.134848,掛茶屋
0.128824,ところ
0.126257,不思議
0.115584,墓参り
0.109669,サルト
0.109669,アンドレア
0.107796,御母さん
0.107796,婆さん

PCA2
0.143961,サルト
0.143961,アンドレア
0.128456,ところ
0.125966,水彩画
0.123357,西洋人
0.123357,掛茶屋
0.117010,御母さん
0.117010,婆さん
0.117010,兄さん
0.105735,墓参り

第1主成分：太宰治軸？
（反芥川軸）

第2主成分：夏目漱石軸？
（反メロス軸）

少ない順

PCA1
-0.254842,芥川龍之介
-0.247133,御釈迦様
-0.247133,血の池
-0.226052,羅生門
-0.169539,髪の毛
-0.161701,自尊心
-0.145716,ファイル
-0.144363,1997
-0.144363,1971
-0.138228,さっき

PCA2
-0.715620,メロス
-0.160073,おまえ
-0.141241,セリヌンティウス
-0.091214,サーヴィス
-0.081008,ひとり
-0.073116,こんど
-0.068411,おなか
-0.068411,これまで
-0.068411,ごはん
-0.068411,セエター

データの次元圧縮

- データを主成分空間に写像

```
>>> x = pca.transform(tfidfs.toarray())
```

データをプロットしてみる

データプロット用ライブラリの読み込み

- matplotlibの読み込み

```
>>> from matplotlib import pyplot
```

- 対話モード

```
>>> pyplot.ion()
```

- プロット面のクリア

```
>>> pyplot.clf()
```

次元圧縮後のデータをプロットする

- 名前の設定

```
>>> name = ['hana', 'kokoro', 'kumo', 'merosu', 'ningen', 'rasho',
'sorekara', 'waganeko']
```

- 色の設定

```
>>> colors = [pyplot.cm.hsv(0.1 * i,1) for i in range(len(name))]
```

- データのプロット

```
>>> for i in range(len(name)):
...     pyplot.scatter(x[i,0], x[i,1], c=colors[i], label=name[i])
```

- 凡例の表示

```
>>> pyplot.legend(loc = 'lower left')
```

プロット例

- 夏目・太宰・芥川の作品が
  グループ分けされている
- 特定のキーワードの影響の可能性
  （メロスとか芥川とか）

- 名詞を除いて分析
  → 文体の分析の可能性