Python ではじめる機械学習

7.3 Bag of Wordsによるテキスト表現

15T4057F　藤井　真

7.3 Bag of Wordsによるテキスト表現

- BoW （Bag of Words）
  - 機械学習で広く用いられているテキストデータ表現
  - シンプル、効率的
  - テキスト内にどの「言葉」が「何回」現れたか

- すももももももものうち
  - すもも　1回
  - も　2回
  - もも　2回
  - の　1回
  - うち　1回

7.3 Bag of Wordsによるテキスト表現

- BoW （Bag of Words）
  - 機械学習で広く用いられているテキストデータ表現
  - シンプル、効率的
  - テキスト内にどの「言葉」が「何回」現れたか

- すももももももものうち
  - すもも　1回
  - も　2回
  - もも　2回
  - の　1回
  - うち　1回

「李も桃も桃のうち」といった
テキストの構造情報は捨てられる

7.3 Bag of Wordsによるテキスト表現

- BoW表現の取得（3ステップ）

①うちの桃と李
②李も桃も桃のうち

1．トークン分割（Tokenization)

個々の文書を単語（トークン）に分割する。
ホワイトスペースや句読点で句切る。

7.3 Bag of Wordsによるテキスト表現

- BoW表現の取得（3ステップ）

①うちの桃と李
②李も桃も桃のうち

1．トークン分割（Tokenization)

個々の文書を単語（トークン）に分割する。
ホワイトスペースや句読点で句切る。

①うち,の,桃,と,李
②李,も,桃,も,桃,の,うち

7.3 Bag of Wordsによるテキスト表現

- BoW表現の取得（3ステップ）

①うちの桃と李
②李も桃も桃のうち

1．トークン分割（Tokenization)

個々の文書を単語（トークン）に分割する。
ホワイトスペースや句読点で句切る。

①うち,の,桃,と,李
②李,も,桃,も,桃,の,うち

2．ボキャブラリ構築（Vocabulary building）

全ての文書に現れる全ての単語を
ボキャブラリとして集め、番号を付ける。

うち	の	桃	と	李	も
1	2	3	4	5	6

7.3 Bag of Wordsによるテキスト表現

- BoW表現の取得（3ステップ）

①うちの桃と李
②李も桃も桃のうち

1．トークン分割（Tokenization)

個々の文書を単語（トークン）に分割する。
ホワイトスペースや句読点で句切る。

①うち,の,桃,と,李
②李,も,桃,も,桃,の,うち

2．ボキャブラリ構築（Vocabulary building）

全ての文書に現れる全ての単語を
ボキャブラリとして集め、番号を付ける。

うち	の	桃	と	李	も
1	2	3	4	5	6

3．エンコード（Encode)

個々の文書に対して
単語が現れる回数をカウントする。

	うち	の	桃	と	李	も
	1	2	3	4	5	6
①	1	1	1	1	1	0
②	1	1	2	0	1	1

7.3 Bag of Wordsによるテキスト表現

- BoW表現の取得（3ステップ）

①うちの桃と李
②李も桃も桃のうち

1．トークン分割（Tokenization)

個々の文書を単語（トークン）に分割する。
ホワイトスペースや句読点で句切る。

①うち,の,桃,と,李
②李,も,桃,も,桃,の,うち

2．ボキャブラリ構築（Vocabulary building）

全ての文書に現れる全ての単語を
ボキャブラリとして集め、番号を付ける。

うち	の	桃	と	李	も
1	2	3	4	5	6

3．エンコード（Encode)

個々の文書に対して
単語が現れる回数をカウントする。

「李も桃も桃のうち」といった
テキストの構造情報は捨てられる

	うち	の	桃	と	李	も
	1	2	3	4	5	6
①	1	1	1	1	1	0
②	1	1	2	0	1	1

7.3.1 トイデータセットに対するBoW

- BoW表現は変換器CountVectorizerを用いる

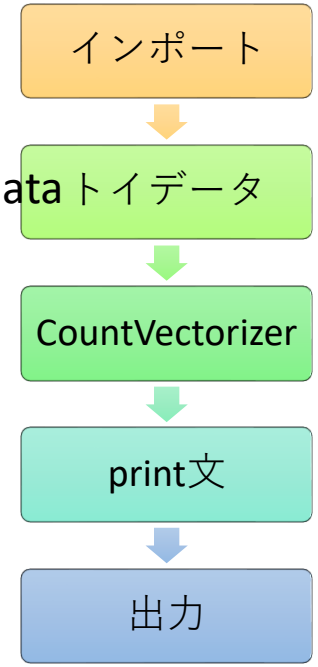
```
from sklearn.feature_extraction.text import CountVectorizer

peach_words =["uti no sumomo to momo",
              "sumomo mo momo mo momo no uti"]

vect = CountVectorizer()
vect.fit(peach_words)

print("Vocabulary size: {}".format(len(vect.vocabulary_)))
print("Vocabulary content:\n {}".format(vect.vocabulary_))
```

```
Vocabulary size: 6
Vocabulary content:
 {'uti': 5, 'no': 2, 'sumomo': 3, 'to': 4, 'momo': 1, 'mo': 0}
```

おもちゃのデータ
toy data トイデータ

インスタンス生成
データをfit

vocabulary属性を表示



1．トークン分割
2．ボキャブラリ構築
　　が終わった状態

7.3.1 トイデータセットに対するBoW

- CountVectorizerのtransformメソッドでBoWに

```
peach_words_bow = vect.transform(peach_words)
print("peach_words_bow:\n {}".format(repr(peach_words_bow)))
print("Dense representation:\n{}".format(peach_words_bow))
print("Sparse representation:\n{}".format(peach_words_bow.toarray()))
```

```
peach_words_bow:
 <2x6 sparse matrix of type '<class 'numpy.int64'>'
        with 10 stored elements in Compressed Sparse Row format>
Dense representation:
  (0, 1)        1
  (0, 2)        1
  (0, 3)        1
  (0, 4)        1
  (0, 5)        1
  (1, 0)        2
  (1, 1)        2
  (1, 2)        1
  (1, 3)        1
  (1, 5)        1
Sparse representation:
[[0 1 1 1 1 1]
 [2 2 1 1 0 1]]
```

２×６の疎行列、要素１０

標準状態では密行列形式で格納されている。

※トイデータがあまりに小さいので疎密が逆に見える
本来的なデータを対象にすると疎行列で表示しきれない

toarray()メソッドで疎行列として表示

０：mo　１：momoなので登場回数は合っている

7.3.2 映画レビューのBoW

- 映画レビューのセンチメント分析タスクに適用
  - 7.2節でデータセットの詳細は見たので概略
    - 映画レビューが全部で50,000
    - そのうち訓練用25,000、テスト用25,000
    - その各々にpostiveデータ12,500、negativeデータ12,500
    - 訓練でpos/negの判別モデルを作り、テストする

  - 1レビュー例

```
print("text_train[1]:\n{}".format(text_train[1]))
```

```
text_train[1]:
b'Words can\'t describe how bad this movie is. I can\'t explain it by writing only. You have
too see it for yourself to get at grip of how horrible a movie really can be. Not that I reco
mmend you to do that. There are so many clich\xc3\xa9s, mistakes (and all other negative thin
gs you can imagine) here that will just make you cry. To start with the technical first, ther
e are a LOT of mistakes regarding the airplane. I won\'t list them here, but just mention the
coloring of the plane. They didn\'t even manage to show an airliner in the colors of a fictio
nal airline, but instead used a 747 painted in the original Boeing livery. Very bad. The plot
is stupid and has been done many times before, only much, much better. There are so many ridi
culous moments here that i lost count of it really early. Also, I was on the bad guys\' side
all the time in the movie, because the good guys were so stupid. "Executive Decision" should
without a doubt be you\'re choice over this one, even the "Turbulence"-movies are better. In
fact, every other movie in the world is better than this one.'
```

7.3.2 映画レビューのBoW

- データ読込、BoW表現の構築

```
from sklearn.datasets import load_files

reviews_train = load_files("aclImdb/train/")
reviews_test  = load_files("aclImdb/test/")
text_train, y_train = reviews_train.data, reviews_train.target
text_test,  y_test  = reviews_test.data,  reviews_test.target
print("type   of text_train: {}".format(type(text_train)))
print("length of text_train: {}".format(len(text_train)))
print("length of text_test : {}".format(len(text_test)))
```

データのロード

データの切り分け

```
type   of text_train: <class 'list'>
length of text_train: 25000
length of text_test : 25000
```

```
vect = CountVectorizer().fit(text_train)
X_train = vect.transform(text_train)
print("X_train:\n{}".format(repr(X_train)))
```

CountVectorizer、fit
transform（BoW表現）

```
X_train:
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'
        with 3445861 stored elements in Compressed Sparse Row format>
```

２５,０００×７４,８４９の疎行列、要素３,４４５,８６１

7.3.2 映画レビューのBoW

- ２５,０００×７４,８４９の疎行列を覗く
  - get_feature_names()メソッド：特徴量のリスト

```
feature_names = vect.get_feature_names()
print("Number of features: {}".format(len(feature_names)))
print("First 20 features:\n{}".format(feature_names[:20]))
print("Features 20010 to 20020:\n{}".format(feature_names[20010:20020]))
print("Every 5000th feature:\n{}".format(feature_names[::5000]))
```

リストの長さ

```
Number of features: 74849
First 20 features:
['00', '000', '0000000000001', '00001', '00015', '000s', '001', '003830', '006', '007', '007
9', '0080', '0083', '0093638', '00am', '00pm', '00s', '01', '01pm', '02']
Features 20010 to 20020:
['dratted', 'draub', 'draught', 'draughts', 'draughtswoman', 'draw', 'drawback', 'drawbacks',
'drawer', 'drawers']
Every 5000th feature:
['00', 'augustine', 'bête', 'cost', 'draper', 'fleece', 'hasan', 'jardine', 'maars', 'nathani
el', 'pincher', 'replica', 'shunning', 'swordmen', 'unproven']
```

最初の２０

２００１０から２００２０

５０００毎

  - ‘000000000001’や’00015’、’003830’など一回きりであろう数値がpos/negに深く関わるだろうか。
  - ‘007’は分かる人には分かるフレーズだが、意味の無い数値と思う人もいるだろう。

7.3.2 映画レビューのBoW

- LogisticRegressionを元にした分析
  - まずは交差検証とグリッドサーチでモデル生成

```
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
scores = cross_val_score(LogisticRegression(), X_train, y_train, cv=5)
print("Mean cross-validation accuracy: {:.2f}".format(np.mean(scores)))
```

```
Mean cross-validation accuracy: 0.88
```

交差検証の平均スコア

```
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
print("Best parameters: ", grid.best_params_)
```

```
Best cross-validation score: 0.89
Best parameters:  {'C': 0.1}
```

線形回帰のCパラメータをグリッドサーチ

7.3.2 映画レビューのBoW

- 作成したモデルでテスト

```
X_test = vect.transform(text_test)
print("Test score: {:.2f}".format(grid.score(X_test, y_test)))
```

```
Test score: 0.88
```

7.3.2 映画レビューのBoW

- 単語の抽出方法へのアプローチ
  - min_dfパラメータで抽出されるための条件を設定

```
print("X_train:\n{}".format(repr(X_train)))

vect = CountVectorizer(min_df=5).fit(text_train)
X_train = vect.transform(text_train)
print("X_train with min_df:\n {}".format(repr(X_train)))
```

```
X_train:
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'
        with 3445861 stored elements in Compressed Sparse Row format>
X_train with min_df:
 <25000x27272 sparse matrix of type '<class 'numpy.int64'>'
        with 3368680 stored elements in Compressed Sparse Row format>
```

  - 5文書以上の登場を条件にする
  - 特徴量数74,849⇒27,272
  - 要素数3,445,861⇒3,368,680

7.3.2 映画レビューのBoW

- ２５,０００×２７,２７２の疎行列を覗く
  - get_feature_names()メソッド：特徴量のリスト

```
feature_names = vect.get_feature_names()
print("Number of features: {}".format(len(feature_names)))
print("First 20 features:\n{}".format(feature_names[:20]))
print("Features 20010 to 20020:\n{}".format(feature_names[20010:20020]))
print("Every 5000th feature:\n{}".format(feature_names[::5000]))
```

```
Number of features: 27272
First 20 features:
['00', '000', '007', '00s', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '10
0', '1000', '100th', '101', '102', '103']
Features 20010 to 20020:
['repent', 'repentance', 'repercussions', 'repertoire', 'repetition', 'repetitions', 'repetit
ious', 'repetitive', 'rephrase', 'replace']
Every 5000th feature:
['00', 'complicit', 'gainey', 'martians', 'repairs', 'trenches']
```

最初の２０

２００１０から２００２０

５０００毎

- ‘000000000001’や’00015’、’003830’など一回きりであろう数値が 消えた。
- ‘007’など一見無意味な数値も、5文書以上で現れたため無意味と思えないデータとして残っている。

※『007』は映画のタイトル

7.3.2 映画レビューのBoW

- グリッドサーチスコア

```
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

```
Best cross-validation score: 0.89
```

- 結果は０．８８⇒０．８９　劇的変化なし
- ただし、特徴量数74,849⇒27,272の恩恵として
  高速化している。速くなって精度そのまま。

…ちなみに
テストデータの中に訓練データに無い単語が現れても無視される。
その多くは登場回数が少ないことが予想され、影響が小さいため。

7.3 Bag of Wordsによるテキスト表現

- まとめ
  - BoW表現は3ステップ
    - トークン分割・ボキャブラリ構築・エンコード
  - 文法構造の情報は捨てられる。
  - Scikit-learnではCountVectorizerとtransformメソッド。
  - 特徴量抽出は高速化の面だけ見ても価値あり。

	条件なし	5文書以上現れる単語
単語数	74849	27272
要素数	3445861	3368680
交差検証スコア	0.88	0.89
最初の20	'00', '000', '0000000000001', '00001', '00015', '000s', '001', '003830', '006', '007', '0079', '0080', '0083', '0093638', '00am', '00pm', '00s', '01', '01pm', '02'	'00', '000', '007', '00s', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '100', '1000', '100th', '101', '102', '103'
20010-20020	'dratted', 'draub', 'draught', 'draughts', 'draughtswoman', 'draw', 'drawback', 'drawbacks', 'drawer', 'drawers'	'repent', 'repentance', 'repercussions', 'repertoire', 'repetition', 'repetitions', 'repetitious', 'repetitive', 'rephrase', 'replace'
5000毎	'00', 'augustine', 'bête', 'cost', 'draper', 'fleece', 'hasan', 'jardine', 'maars', 'nathaniel', 'pincher', 'replica', 'shunning', 'swordmen', 'unproven'	'00', 'complicit', 'gainey', 'martians', 'repairs', 'trenches'