

Simple topic identification

Puteaux, Fall/Winter 2020-2021

```
#####  
##                                     ##  
## Natural Language Processing in Python ##  
##                                     ##  
#####
```

\$1 Introduction to Natural Language Processing in Python

\$1.2 Simple topic identification

1 Word counts with bag-of-words

1.1 What is bag-of-words?

- It is a basic method for finding topics in a text.
- Need first to create tokens using tokenization.
- And then count up all the tokens.
- The more frequent a word, the more important it might be.
- It can be a great way to determine the significant words in a text.

1.2 Code of bag-of-words in Python:

```
[1]: from nltk.tokenize import word_tokenize  
from collections import Counter  
  
Counter(  
    word_tokenize("""The cat is in the box. The cat likes the box. \  
The box is over the cat."""))
```

```
[1]: Counter({'The': 3,  
            'cat': 3,  
            'is': 2,  
            'in': 1,  
            'the': 3,  
            'box': 3,  
            '.': 3,
```

```
'likes': 1,
'over': 1})
```

```
[2]: counter = Counter(
      word_tokenize("""The cat is in the box. The cat likes the box. \
The box is over the cat."""))
counter.most_common(2)
```

```
[2]: [('The', 3), ('cat', 3)]
```

1.3 Practice question for bag-of-words picker:

- It's time for a quick check on the understanding of bag-of-words. Which of the below options, with basic NLTK tokenization, map the bag-of-words for the following text?

“The cat is in the box. The cat box.”

- ☐ ('the', 3), ('box.', 2), ('cat', 2), ('is', 1).
- ☐ ('The', 3), ('box', 2), ('cat', 2), ('is', 1), ('in', 1), ('.', 1).
- ☐ ('the', 3), ('cat box', 1), ('cat', 1), ('box', 1), ('is', 1), ('in', 1).
- ☒ ('The', 2), ('box', 2), ('.', 2), ('cat', 2), ('is', 1), ('in', 1), ('the', 1).

► Question-solving method:

```
[3]: from nltk.tokenize import word_tokenize
      from collections import Counter

      Counter(word_tokenize("The cat is in the box. The cat box."))
```

```
[3]: Counter({'The': 2, 'cat': 2, 'is': 1, 'in': 1, 'the': 1, 'box': 2, '.': 2})
```

1.4 Practice exercises for word counts with bag-of-words:

► Package pre-loading:

```
[4]: from nltk import word_tokenize
```

► Data pre-loading:

```
[5]: article = open('ref1. Wikipedia article - Debugging.txt').read()
```

► Bag-of-words Counter building practice:

```
[6]: # Import Counter
      from collections import Counter

      # Tokenize the article: tokens
```

```
tokens = word_tokenize(article)

# Convert the tokens into lowercase: lower_tokens
lower_tokens = [t.lower() for t in tokens]

# Create a Counter with the lowercase tokens: bow_simple
bow_simple = Counter(lower_tokens)

# Print the 10 most common tokens
print(bow_simple.most_common(10))
```

```
[(',', 151), ('the', 150), ('.', 89), ('of', 81), ('"', 66), ('to', 63), ('a', 60), ('`', 47), ('in', 44), ('and', 41)]
```

```
#####
##                                     ##
##  Natural Language Processing in Python  ##
##                                     ##
#####

$1 Introduction to Natural Language Processing in Python
$1.2 Simple topic identification
```

2 Simple text preprocessing

2.1 Why preprocess?

- When performing machine learning or other statistical methods, it could help make for better input data.
- Examples:
 - *tokenization to create a bag of words*
 - *lowercasing words*
- Lemmatization/Stemming:
 - shorten words to their root stems
- Remove stop words, punctuation, or unwanted tokens.
- Good to experiment with different approaches.

2.2 Code of text preprocessing with Python:

```
[7]: from nltk.tokenize import word_tokenize
      from collections import Counter
```

```
[8]: from nltk.corpus import stopwords

text = """The cat is in the box. The cat likes the box.
The box is over the cat."""
tokens = [w for w in word_tokenize(text.lower()) if w.isalpha()]
no_stops = [t for t in tokens if t not in stopwords.words('english')]
Counter(no_stops).most_common(2)
```

```
[8]: [('cat', 3), ('box', 3)]
```

```
[9]: from nltk.stem import WordNetLemmatizer

text = """Cats, dogs and birds are common pets. So are fish."""
tokens = [w for w in word_tokenize(text.lower()) if w.isalpha()]
no_stops = [t for t in tokens if t not in stopwords.words('english')]
wordnet_lemmatizer = WordNetLemmatizer()
lemmatized = [wordnet_lemmatizer.lemmatize(t) for t in no_stops]
print(lemmatized)
```

```
['cat', 'dog', 'bird', 'common', 'pet', 'fish']
```

2.3 Practice question for text preprocessing steps:

- Which of the following are useful text preprocessing steps?
 - ☐ Stems, spelling corrections, lowercase.
 - ☒ Lemmatization, lowercasing, removing unwanted tokens.
 - ☐ Removing stop words, leaving in capital words.
 - ☐ Strip stop words, word endings and digits.

2.4 Practice exercises for simple text preprocessing:

► Package pre-loading:

```
[10]: from nltk import word_tokenize
from collections import Counter
```

► Data pre-loading:

```
[11]: article = open('ref1. Wikipedia article - Debugging.txt').read()
tokens = word_tokenize(article)
lower_tokens = [t.lower() for t in tokens]
stopwords = open('ref2. English stopwords.txt').read()
english_stops = word_tokenize(stopwords)
```

► Text preprocessing practice:

```
[12]: # Import WordNetLemmatizer
from nltk.stem import WordNetLemmatizer

# Retain alphabetic words: alpha_only
alpha_only = [t for t in lower_tokens if t.isalpha()]

# Remove all stop words: no_stops
no_stops = [t for t in alpha_only if t not in english_stops]

# Instantiate the WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

# Lemmatize all tokens into a new list: lemmatized
lemmatized = [wordnet_lemmatizer.lemmatize(t) for t in no_stops]

# Create the bag-of-words: bow
bow = Counter(lemmatized)

# Print the 10 most common tokens
print(bow.most_common(10))

[('debugging', 40), ('system', 25), ('bug', 17), ('software', 16), ('problem',
15), ('tool', 15), ('computer', 14), ('process', 13), ('term', 13), ('debugger',
13)]

#####
##                               ##
##  Natural Language Processing in Python  ##
##                               ##
#####

$1 Introduction to Natural Language Processing in Python

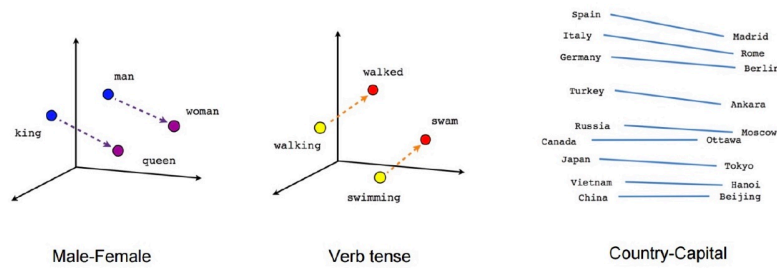
$1.2 Simple topic identification
```

3 Introduction to gensim

3.1 What is gensim?

- It is a popular open-source NLP library.
- It uses top academic models to perform complex tasks:
 - building document or word vectors
 - performing topic identification and document comparison

3.2 What is a word vector?



3.3 Code of creating a gensim corpus:

```
[13]: from gensim.corpora.dictionary import Dictionary
      from nltk.tokenize import word_tokenize
```

```
my_documents = [
    'The movie was about a spaceship and aliens.',
    'I really liked the movie!',
    'Awesome action scenes, but boring characters.',
    'The movie was awful! I hate alien films.',
    'Space is cool! I liked the movie.',
    'More space films, please!',
]
```

```
[14]: tokenized_docs = [word_tokenize(doc.lower()) for doc in my_documents]
```

```
dictionary = Dictionary(tokenized_docs)
dictionary.token2id
```

```
[14]: {'.': 0,
      'a': 1,
      'about': 2,
      'aliens': 3,
      'and': 4,
      'movie': 5,
      'spaceship': 6,
      'the': 7,
      'was': 8,
      '!': 9,
      'i': 10,
      'liked': 11,
      'really': 12,
      ',': 13,
      'action': 14,
      'awesome': 15,
      'boring': 16,
```

```
'but': 17,  
'characters': 18,  
'scenes': 19,  
'alien': 20,  
'awful': 21,  
'films': 22,  
'hate': 23,  
'cool': 24,  
'is': 25,  
'space': 26,  
'more': 27,  
'please': 28}
```

```
[15]: corpus = [dictionary.doc2bow(doc) for doc in tokenized_docs]  
corpus
```

```
[15]: [[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1)],  
      [(5, 1), (7, 1), (9, 1), (10, 1), (11, 1), (12, 1)],  
      [(0, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18, 1), (19, 1)],  
      [(0, 1),  
       (5, 1),  
       (7, 1),  
       (8, 1),  
       (9, 1),  
       (10, 1),  
       (20, 1),  
       (21, 1),  
       (22, 1),  
       (23, 1)],  
      [(0, 1), (5, 1), (7, 1), (9, 1), (10, 1), (11, 1), (24, 1), (25, 1), (26, 1)],  
      [(9, 1), (13, 1), (22, 1), (26, 1), (27, 1), (28, 1)]]
```

3.4 What are the advantages of creating a gensim corpus?

- First of all, gensim models can be easily saved, updated, and reused.
- Secondly, the dictionary created can also be updated.
- Lastly, the more advanced and feature-rich bag-of-words can be used in future exercises.

3.5 Practice question for word vectors:

- What are word vectors, and how do they help with NLP?
 - ☐ They are similar to bags of words, just with numbers. You use them to count how many tokens there are.
 - ☐ Word vectors are sparse arrays representing bigrams in the corpora. You can use them to compare two sets of words to one another.

- ☒ Word vectors are multi-dimensional mathematical representations of words created using deep learning methods. They give us insight into relationships between words in a corpus.
- ☐ Word vectors don't actually help NLP and are just hype.

3.6 Practice exercises for introduction to gensim:

► Package pre-loading:

```
[16]: import zipfile

from nltk import word_tokenize
```

► Data pre-loading:

```
[17]: file_name = 'ref4. Wikipedia articles.zip'
with zipfile.ZipFile(file_name, 'r') as archive:
    files = [
        archive.read(name) for name in archive.namelist()
        if name.endswith('.txt')
    ]

doc_tokens = [word_tokenize(file.decode("utf-8")) for file in files]

articles = []
stopwords = open('ref2. English stopwords.txt').read()
english_stops = word_tokenize(stopwords)
for i in range(len(doc_tokens)):
    lower_tokens = [t.lower() for t in doc_tokens[i]]
    alphanumeric_only = [t for t in lower_tokens if t.isalnum()]
    no_stops = [t for t in alphanumeric_only if t not in english_stops]
    articles.append(no_stops)
```

► Gensim corpus creating and querying practice:

```
[18]: # Import Dictionary
from gensim.corpora.dictionary import Dictionary

# Create a Dictionary from the articles: dictionary
dictionary = Dictionary(articles)

# Select the id for "computer": computer_id
computer_id = dictionary.token2id.get("computer")

# Use computer_id with the dictionary to print the word
print(dictionary.get(computer_id))

# Create a MmCorpus: corpus
corpus = [dictionary.doc2bow(article) for article in articles]
```



```
# Print the first 10 word ids with their frequency counts from the fifth  
→ document  
print(corpus[4][:10])
```

```
computer  
[(13, 2), (24, 1), (43, 1), (44, 6), (45, 1), (50, 1), (58, 1), (59, 1), (61,  
7), (75, 1)]
```

► Package pre-loading:

```
[19]: from collections import defaultdict  
import itertools
```

► Gensim bag-of-words practice:

```
[20]: # Save the fifth document: doc  
doc = corpus[4]  
  
# Sort the doc for frequency: bow_doc  
bow_doc = sorted(doc, key=lambda w: w[1], reverse=True)  
  
# Print the top 5 words of the document alongside the count  
for word_id, word_count in bow_doc[:5]:  
    print(dictionary.get(word_id), word_count)  
  
# Create the defaultdict: total_word_count  
total_word_count = defaultdict(int)  
for word_id, word_count in itertools.chain.from_iterable(corpus):  
    total_word_count[word_id] += word_count
```

```
language 54  
programming 39  
languages 30  
code 22  
computer 15
```

```
[21]: # Save the fifth document: doc  
doc = corpus[4]  
  
# Sort the doc for frequency: bow_doc  
bow_doc = sorted(doc, key=lambda w: w[1], reverse=True)  
  
# Print the top 5 words of the document alongside the count  
for word_id, word_count in bow_doc[:5]:  
    print(dictionary.get(word_id), word_count)  
  
# Create the defaultdict: total_word_count
```

```
total_word_count = defaultdict(int)
for word_id, word_count in itertools.chain.from_iterable(corpus):
    total_word_count[word_id] += word_count

# Create a sorted list from the defaultdict: sorted_word_count
sorted_word_count = sorted(total_word_count.items(),
                           key=lambda w: w[1],
                           reverse=True)

# Print the top 5 words across all documents alongside the count
for word_id, word_count in sorted_word_count[:5]:
    print(dictionary.get(word_id), word_count)
```

```
language 54
programming 39
languages 30
code 22
computer 15
computer 598
software 450
cite 322
ref 259
code 235
```

```
#####
##                                     ##
##  Natural Language Processing in Python  ##
##                                     ##
#####
```

```
$1 Introduction to Natural Language Processing in Python
```

```
$1.2 Simple topic identification
```

4 Tf-idf with gensim

4.1 What is tf-idf?

- Tf-idf means term frequency - inverse document frequency.
- Allow determining the most important words in each document.
- Each corpus may have shared words beyond just stopwords.
- These words should be down-weighted in importance.
- Example:
 - “sky” from the theme of astronomy
- Ensures most common words don’t show up as keywords.

- Keep document specific frequent words weighted high.

4.2 What is the tf-idf formula?

- $w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$
 - $w_{i,j}$ = tf-idf weight for token i in document j
 - $tf_{i,j}$ = number of occurrences of token i in document j
 - df_i = number of documents that contain token i
 - N = total number of documents

4.3 Code of tf-idf with gensim:

```
[22]: from gensim.corpora.dictionary import Dictionary
      from nltk.tokenize import word_tokenize

      my_documents = [
          'The movie was about a spaceship and aliens.',
          'I really liked the movie!',
          'Awesome action scenes, but boring characters.',
          'The movie was awful! I hate alien films.',
          'Space is cool! I liked the movie.',
          'More space films, please!',
      ]

      tokenized_docs = [word_tokenize(doc.lower()) for doc in my_documents]
      dictionary = Dictionary(tokenized_docs)
      corpus = [dictionary.doc2bow(doc) for doc in tokenized_docs]
```

```
[23]: from gensim.models.tfidfmodel import TfidfModel

      tfidf = TfidfModel(corpus)
      tfidf[corpus[1]]
```

```
[23]: [(5, 0.1746298276735174),
      (7, 0.1746298276735174),
      (9, 0.1746298276735174),
      (10, 0.29853166221463673),
      (11, 0.47316148988815415),
      (12, 0.7716931521027908)]
```

4.4 Practice question for what is tf-idf:

- To calculate the tf-idf weight for the word “computer”, which appears five times in a document containing 100 words. Given a corpus containing 200 documents, with 20 documents mentioning the word “computer”, so tf-idf can be calculated by multiplying term frequency with inverse document frequency.

- Notes:
 - term frequency = percentage share of the word compared to all tokens in the document
 - inverse document frequency = logarithm of the total number of documents in a corpus divided by the number of documents containing the term
- Which of the below options is correct?
 - ☒ $(5 / 100) * \log(200 / 20)$
 - ☐ $(5 * 100) / \log(200 * 20)$
 - ☐ $(20 / 5) * \log(200 / 20)$
 - ☐ $(200 * 5) * \log(400 / 5)$

4.5 Practice exercises for tf-idf with gensim:

► Package pre-loading:

```
[24]: import zipfile

from nltk import word_tokenize

from gensim.corpora.dictionary import Dictionary
from gensim.models.tfidfmodel import TfidfModel
```

► Data pre-loading:

```
[25]: file_name = 'ref4. Wikipedia articles.zip'
with zipfile.ZipFile(file_name, 'r') as archive:
    files = [
        archive.read(name) for name in archive.namelist()
        if name.endswith('.txt')
    ]

doc_tokens = [word_tokenize(file.decode("utf-8")) for file in files]
articles = []
stopwords = open('ref2. English stopwords.txt').read()
english_stops = word_tokenize(stopwords)
for i in range(len(doc_tokens)):
    lower_tokens = [t.lower() for t in doc_tokens[i]]
    alphanumeric_only = [t for t in lower_tokens if t.isalnum()]
    no_stops = [t for t in alphanumeric_only if t not in english_stops]
    articles.append(no_stops)

dictionary = Dictionary(articles)
corpus = [dictionary.doc2bow(article) for article in articles]

doc = corpus[4]
```

► Wikipedia tf-idf practice:

```
[26]: # Create a new TfidfModel using the corpus: tfidf
tfidf = TfidfModel(corpus)

# Calculate the tfidf weights of doc: tfidf_weights
tfidf_weights = tfidf[doc]

# Print the first five weights
print(tfidf_weights[:5])

[(13, 0.021411676334320492), (24, 0.01738903055915624), (43,
0.00805356588388867), (45, 0.021821227698039212), (50, 0.01376766181415054)]

[27]: # Create a new TfidfModel using the corpus: tfidf
tfidf = TfidfModel(corpus)

# Calculate the tfidf weights of doc: tfidf_weights
tfidf_weights = tfidf[doc]

# Print the first five weights
print(tfidf_weights[:5])

# Sort the weights from highest to lowest: sorted_tfidf_weights
sorted_tfidf_weights = sorted(tfidf_weights, key=lambda w: w[1], reverse=True)

# Print the top 5 weighted words
for term_id, weight in sorted_tfidf_weights[:5]:
    print(dictionary.get(term_id), weight)

[(13, 0.021411676334320492), (24, 0.01738903055915624), (43,
0.00805356588388867), (45, 0.021821227698039212), (50, 0.01376766181415054)]
compiled 0.2182122769803921
compilation 0.21353333707313848
eiffel 0.17794444756094874
abstraction 0.1745698215843137
intermediate 0.16521194176980647
```