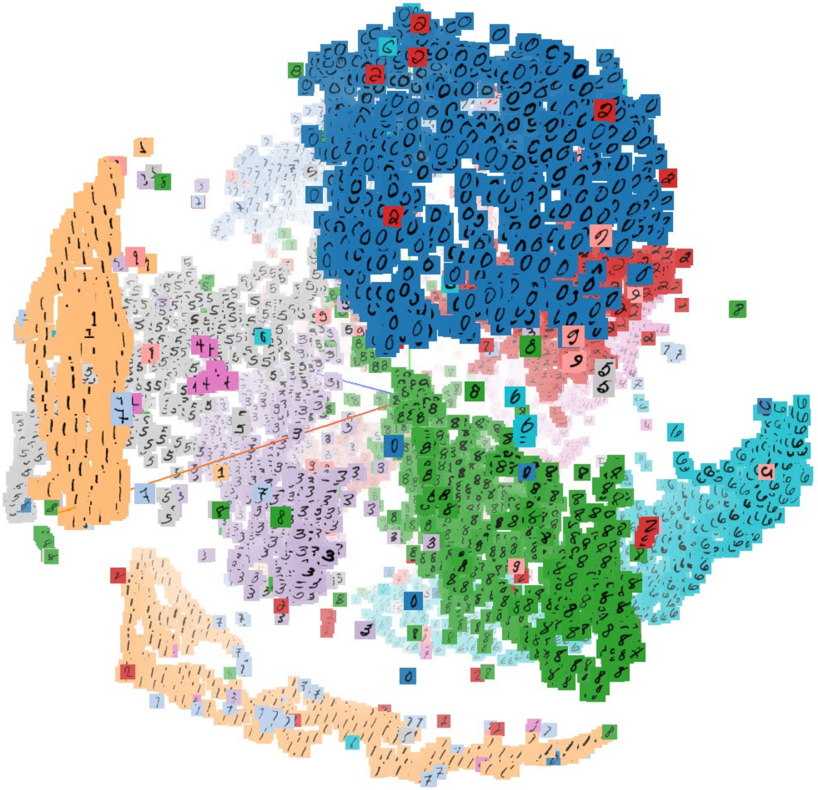
Chapter 3

Simple Topic Identification II

Outline

- Introduction to **gensim**
- **TF-IDF** with gensim

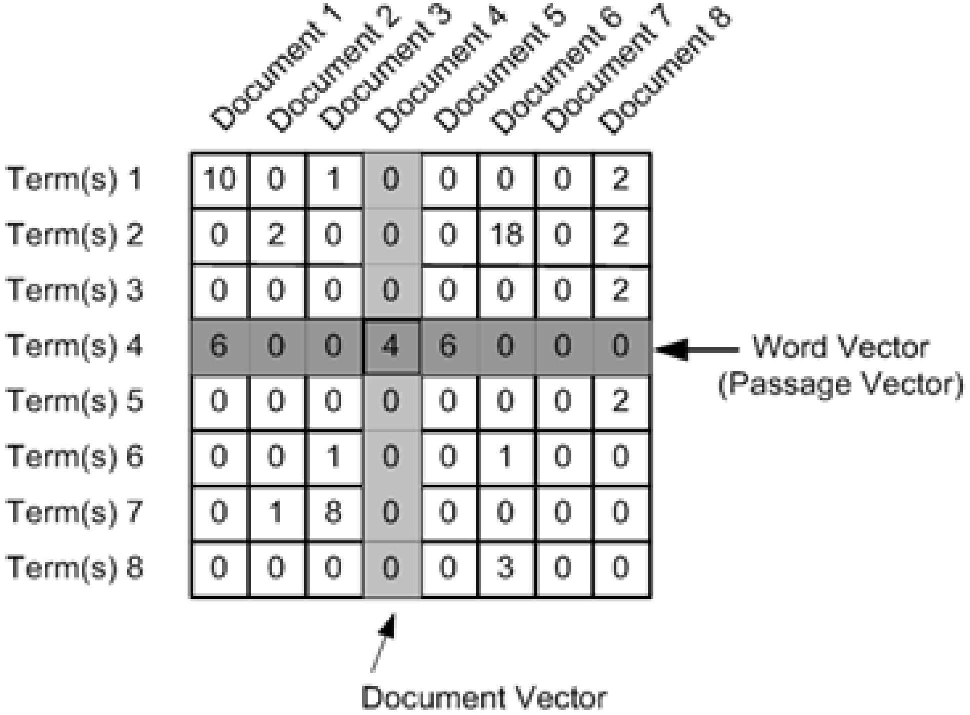


Introduction to gensim

What is gensim?

- A popular open-source natural language processing (NLP) library.
- It uses top academic models to perform complex tasks like
  - building document or word vectors (a vector of weights), corpora and
  - **performing topic identification** and document comparisons.

Word Embeddings or Word vectorization is a methodology in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics. The process of converting words into numbers are called Vectorization.



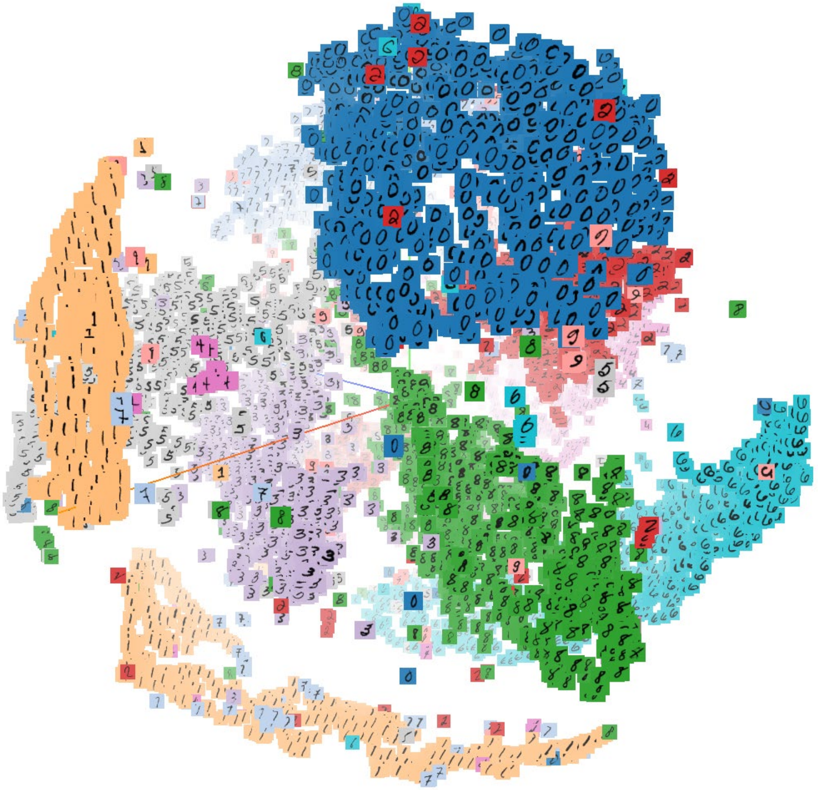
	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

Word Vector (Passage Vector)

Document Vector

Creating a gensim dictionary

- Gensim allows you to build **corpora** and **dictionaries** using simple classes and functions.
- A corpus (or corpora) is **a set of texts** used to help perform natural language processing tasks.



Let’s practice!

Creating and querying a corpus with gensim

- To create gensim dictionary and corpus
- Install gensim

  ```
  pip install gensim
  ```

- Create a list of document tokens called **articles**
  - Select 10 articles
  - Preprocess by lowercasing all words, tokenizing them, removing stop words and punctuation
  - Save them to **articles**

Create a list of document tokens

```
articles = []
for i in range(10) :
    #Read TXT file
    f = open(f".\ch3\wiki\wiki_article_{i}.txt", "r")
    article = f.read()
    # Tokenize the article: tokens
    tokens = _______________(article)
    # Convert the tokens into lowercase: lower_tokens
    lower_tokens = [t._________() for t in tokens]
    # Retain alphabetic words: alpha_only
    alpha_only = [t for t in lower_tokens if t.____________()]
    # Remove all stop words: no_stops
    no_stops = [t for t in alpha_only if t not in __________________________]
    # Instantiate the WordNetLemmatizer
    wordnet_lemmatizer = WordNetLemmatizer()
    # Lemmatize all tokens into a new list: lemmatized
    lemmatized = [wordnet_lemmatizer._________________(t) for t in no_stops]
    #list_article
    articles.append(lemmatized)
print(articles[0])
```

Creating and querying a corpus with gensim

- Import **Dictionary** from **gensim.corpora.dictionary**.

```
# Import Dictionary
from ________________________ import _____________
```

- Initialize a **gensim Dictionary** with the tokens in **articles**.

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

Creating and querying a corpus with gensim

- Obtain the id for **"computer"** from **dictionary**.
  - To do this, use its **.token2id** method which returns ids from text, and then chain .**get()** which returns tokens from ids.
  - Pass in **"computer"** as an argument to .**get()**.

```
# Select the id for "computer": computer_id
computer_id = dictionary._____________.___(“__________")

# Use computer_id with the dictionary to print the word
print(dictionary.____(_____________))
```

Creating and querying a corpus with gensim

- Use a list comprehension in which you iterate over **articles** to create a gensim **corpus** from **dictionary**.
- In the output expression, use the **.doc2bow()** method on **dictionary** with **article** as the argument.

```
# Create a Corpus: corpus
corpus = [dictionary.___________(__) for a in articles]
```

Gensim bag-of-words

- Import **defaultdict** from **collections**.

```
from collections import defaultdict
```

- The Python **defaultdict** and **itertools** to help with the creation of intermediate data structures for analysis.

```
# Save the second document: doc
doc = corpus[1]

# Sort the doc for frequency: bow_doc
bow_doc = sorted(doc, key=lambda w: w[1], reverse=True)
```

Gensim bag-of-words

- Using the first **for** loop, print the top five words of **bow_doc** using each **word_id** with the **dictionary** alongside **word_count**.
  - The **word_id** can be accessed using the **.get()** method of **dictionary**.

```
# Print the top 5 words of the document alongside the count
for word_id, word_count in bow_doc[___]:
    print(dictionary.get(word_id), word_count)
```

Gensim bag-of-words

- Create a **defaultdict** called **total_word_count** in which the keys are all the token ids (**word_id**) and the values are the sum of their occurrence across all documents (**word_count**).
  - to specify **int** when creating the **defaultdict**, and inside the **for** loop, increment each **word_id** of **total_word_count** by **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
```

Gensim bag-of-words

- Create a sorted list from the **defaultdict**, using words across the entire corpus. Use the **.items()** method on **total_word_count** inside **sorted()**.

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

- Similar to how you printed the top five words of **bow_doc** earlier, print the top five words of **sorted_word_count** as well as the number of occurrences of each word across **all the documents**.

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

TF-IDF with gensim

What is TF-IDF

- TF-IDF: term-frequncy - inverse document frequency
- Allows you to determine the most important words in each document in the corpus.
- The idea behind tf-idf is that each corpus might have more shared words than stopwords.
  - These words should be down-weighted in importance.
  - For example, if I am an astronomer, sky might be used often.
- Ensures the most common words don't show up as keywords.
- Keeps the document-specific frequent words weighted high and the common words across the entire corpus weighted low.

TF-IDF formula

$$w_{i,j} = tf_{i,j} * \ log\ (\frac{N}{df_i})$$

where

- $w_{i,j}$ is TF-IDF weight for token i in document j
- $tf_{i,j}$ is the number of occurrences of token i in document j
- df_i is the number of documents that contain token i
- N is the total number of documents

TF-IDF with gensim

```
from gensim.models.tfidfmodel import TfidfModel
tfidf = TfidfModel(corpus)
print(tfidf[corpus[0]])
```

```
[ (0, 0.04637388957601683),
  (1, 0.04637388957601683),
  (2, 0.04637388957601683),
  (3, 0.04637388957601683),
  (5, 0.04637388957601683),
  …
  (33, 0.4637388957601683),
  … ]
```

- Builds a TF-IDF model using Gensim and the corpus which is developed before.
- For the first document in the corpora, we see the token weights along with the token ids.
  - Token id 33 has a weight of 0.46 whereas tokens 0-5 have weights below 0.05.
- These weights help to determine good **topics** and **keywords** for a corpus.

Let’s practice!

What is TF-IDF?

- You want to calculate the TF-IDF weight for the word "computer", which appears 5 times in a document containing 100 words. Given a corpus containing 200 documents, with 20 documents mentioning the word "computer", TF-IDF can be calculated by multiplying term frequency with inverse document frequency.
- Which of the below options is correct?
  a) (5 / 100) * log(200 / 20)
  b) (5 * 100) / log(200 * 20)
  c) (20 / 5) * log(200 / 20)
  d) (200 * 5) * log(400 / 5)

Tf-idf with Wikipedia

- Accesses to the same corpus and dictionary objects that you created in the previous exercises - dictionary, corpus, and doc.
- Import **TfidfModel** from **gensim.models.tfidfmodel**.

```
from gensim.models.tfidfmodel import TfidfModel
```

```
articles = []
for i in range(10) :
    #Read TXT file
    f = open(f".\ch3\wiki\wiki_article_{i}.txt", "r")
    article = f.read()
    # Tokenize the article: tokens
    tokens = ________________(article)
    # Convert the tokens into lowercase: lower_tokens
    lower_tokens = [t.__________() for t in tokens]
    # Retain alphabetic words: alpha_only
    alpha_only = [t for t in lower_tokens if t.______________()]
    # Remove all stop words: no_stops
    no_stops = [t for t in alpha_only if t not in ____________________________]
    # Instantiate the WordNetLemmatizer
    wordnet_lemmatizer = WordNetLemmatizer()
    # Lemmatize all tokens into a new list: lemmatized
    lemmatized = [wordnet_lemmatizer._____________(t) for t in no_stops]
    #list_article
    articles.append(lemmatized)

# Create a Dictionary from the articles: dictionary
dictionary = Dictionary(articles)
# Create a Corpus: corpus
corpus = [dictionary.__________(a) for a in articles]
# Save the second document: doc
doc = corpus[1]
```

Tf-idf with Wikipedia

- Initialize a new **TfidfModel** called **tfidf** using **corpus**.

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

- Use **doc** to calculate the weights by passing **[doc]** to **tfidf**.

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

- Print the first five term ids with weights.

```
# Print the first five weights
print(tfidf_weights[___])
```

```
[(0, 0.0463738895760168З),
(1, 0.0463738895760168З),
(2, 0.0463738895760168З),
(3, 0.0463738895760168З),
(5, 0.0463738895760168З)]
```

Tf-idf with Wikipedia

- Sort the term ids and weights in a new list from highest to lowest weight.

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

- Using your pre-existing **dictionary**, print the top five weighted words (**term_id**) from **sorted_tfidf_weights**, along with their weighted score (**weight**).

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

```
device 0.4637388957601683
operation 0.27824333745610097
like 0.2318694478800841
early 0.18549555830406733
calculation 0.13912166872805048
```

Questions

Reference: https://app.datacamp.com/learn