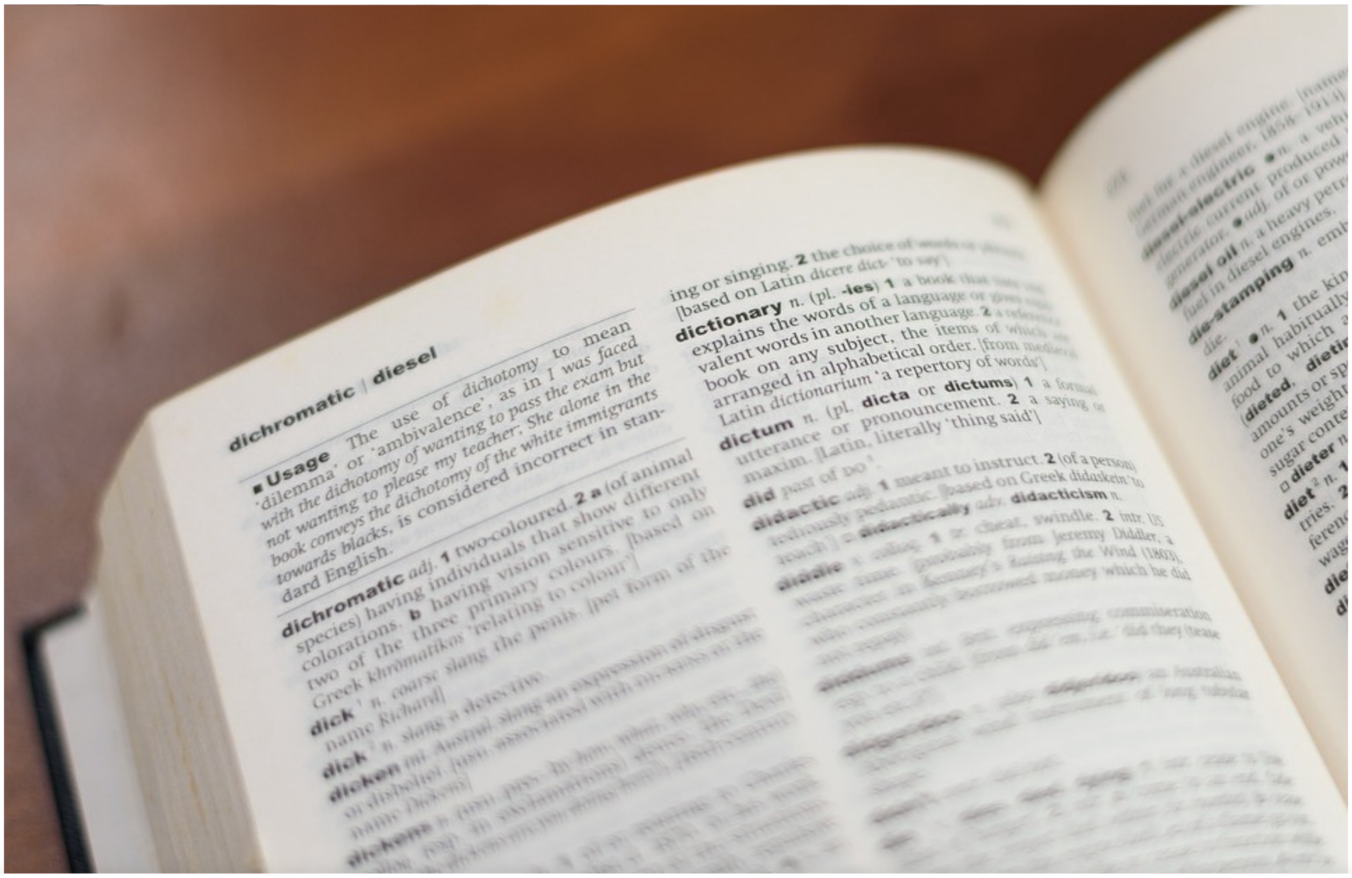
Riley Huang Follow
Jul 10, 2019 · 6 min read ★ · Listen

Calculating Text Similarity With Gensim

Easily comparing different texts with the Genism tool



different attributes of a text file — such as word frequencies, stem words frequencies, sentence lengths, punctuations, and etc. — and compare them in a relatively efficient way.

Thanks to this Python project, I was assigned to do a similar project alone during my summer internship. I was told to build a tool that compares a large number of films and television products with their different attributes. The tool was supposed to generate a set of dictionaries with the same or the most similar products in pairs with different similarity scores.

I started doing the project by using the same approach I took last time, but it didn't go well. Then, I was introduced to **Gensim**.

The content on Gensim's official website and its tutorial are a little too vague, so I wanted to talk a bit about what is Gensim. More importantly, I want to talk about my experience of learning and using Gensim.

GENSIM

topic modelling for humans

> Gensim is a free Python library designed to automatically extract semantic topics from documents, as efficiently (computer-wise) and painlessly (human-wise) as possible. Gensim is designed to process raw, unstructured digital texts ("plain text").

Main Concept

Corpus

The corpus is used to train a machine learning model in Gensim, and the models use the corpus to initialize parameters for the model.

Vector Space Model (VSM)

Every document is represented by an array of features, and you can think of the feature as a question-answer pair. An example of a feature could be the following: **How many times does the word "happy" appear in the text document? Three.**

The question is represented by its id (integer), and hence the representation of the text document becomes a series of pairs, such as (2, 4.0), (3, 6.0), (4, 5.0). This series can be thought of as a **vector.**

If the vectors in the two documents are similar, the documents must be similar too.

Sparse Vector

Model

A Model can be thought of as a transformation from one vector space to another. By training the corpus, the parameters of this transformation are learned.

Use Gensim to Determine Text Similarity

Here's a simple example of code implementation that generates text similarity:

(Here, `jieba` is a text segmentation Python module for cutting the words into segmentations for easier analysis of text similarity in the future.)

```
from gensim import corpora, models, similarities
import jieba

texts = ['I love reading Japanese novels. My favorite Japanese
writer is Tanizaki Junichiro.', 'Natsume Soseki is a well-known
Japanese novelist and his Kokoro is a masterpiece.', 'American
modern poetry is good. ']

keyword = 'Japan has some great novelists. Who is your favorite
Japanese writer?'

texts = [jieba.lcut(text) for text in texts]

dictionary = corpora.Dictionary(texts)

feature_cnt = len(dictionary.token2id)

corpus = [dictionary.doc2bow(text) for text in texts]

tfidf = models.TfidfModel(corpus)

kw_vector = dictionary.doc2bow(jieba.lcut(keyword))

index = similarities.SparseMatrixSimilarity(tfidf[corpus],
for i in range(len(sim)):
    print('keyword is similar to text%d: %.2f' % (i + 1, sim[i]))
```

Print Results:

```
keyword is similar to text1: 0.50
keyword is similar to text2: 0.02
keyword is similar to text3: 0.00
```

Code Explanation

Step 1: Word segmentation using Jieba

First, let's look at how `jieba` works. We want to segment a quote from Tanizaki Junichiro's novel ***Naomi****:*

```
import jieba

text = ‘I wanted to boast to everyone. This woman is mine. Take a
look at my treasure. ‘

words = jieba.lcut(text)

print(words)
```

Print result:

```
[‘I’, ‘wanted’, ‘to’, ‘boast’, ‘to’, ‘everyone’, ‘.’, ‘This’,
‘woman’, ‘is’, ‘mine’, ‘.’, ‘Take’, ‘a’, ‘look’, ‘at’, ‘my’,
```

`len(dictionary.token2id)` represents the number of words in the dictionary.

Example:

```
from gensim import corpora
import jieba


text1 = ‘痴人の愛’

text2 = ‘よく世間では「女が男を欺す」と云います。’

texts = [text1, text2]

# Generate a word list for the text set

texts = [jieba.lcut(text) for text in texts]

print(‘Text set:’, texts)

#Build a dictionary based on a text set

dictionary = corpora.Dictionary(texts)

print(‘dictionary:’, dictionary)

# Extract dictionary features

feature_cnt = len(dictionary.token2id)

print(‘Dictionary feature number: %d’ % feature_cnt)
```

Print result:

```
Text set: [['痴人', 'の', '愛'], ['よ', 'く', '世間', 'で', 'は', '「',
'女', 'が', '男', 'を', '欺', 'す', '」', 'と', '云', 'い', 'ま', 'す',
Dictionary feature number: 21
```

Step 3: Obtain corpus based on dictionary

Example:

```
from gensim import corpora
import jieba

text1 = 'I love Tokyo'

text2 = 'Tokyo, Tokyo, Tokyo'

texts = [text1, text2]

texts = [jieba.lcut(text) for text in texts]

dictionary = corpora.Dictionary(texts)

corpus = [dictionary.doc2bow(text) for text in texts]

print('Dictionary (dictionary):', dictionary.token2id)

print('Corpus: ', corpus)
```

Print result:

```
Dictionary (dictionary): {' ': 0, 'Come': 1, 'Tokyo': 2, 'cuisine':
3, 'for': 4, 'to': 5, ',': 6}

Corpus: [[(0, 5), (1, 1), (2, 2), (3, 1), (4, 1), (5, 1)], [(0, 2),
(2, 3), (6, 2)]]
```

Here, the `doc2bow` function generates Sparse Vector.

```
tfidf = models.TfidfModel(corpus)


index = similarities.SparseMatrixSimilarity(tfidf[corpus], num_features =

feature_cnt)
```

Step 5: Convert search words into Sparse Vectors

Example:

```
from gensim import corpora
import jieba

text1 = 'Tanizaki Junichiro writes good stories'

text2 = 'Naomi is a story written by Tanizaki'

texts = [text1, text2]

texts = [jieba.lcut(text) for text in texts]

dictionary = corpora.Dictionary(texts)

# Use [Dictionary] to convert [search word] to [sparse vector]
keyword = 'good stories'

kw_vector = dictionary.doc2bow(jieba.lcut(keyword))

print(kw_vector)
```

Print result:

```
[(0, 1), (3, 1), (4, 1)]
```

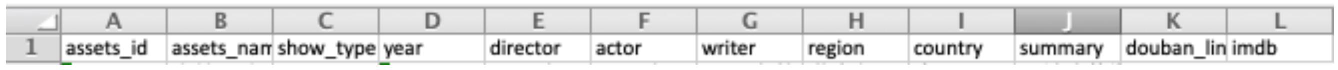
Step 5: Put everything together: similarity calculation

```
feature_cnt = len(dictionary.token2id)

corpus = [dictionary.doc2bow(text) for text in texts]

tfidf = models.TfidfModel(corpus)

kw_vector = dictionary.doc2bow(jieba.lcut(keyword))

index = similarities.SparseMatrixSimilarity(tfidf[corpus],
num_features = feature_cnt)

sim = index[tfidf[kw_vector]]

for i in range(len(sim)):
    print('keyword is similar to text%d: %.2f' % (i + 1, sim[i]))
```

Use of Gensim in the Film Comparison Project

After learning the basic uses of Gensim, I incorporated Gensim in my project to compare film and television.

First of all, the very first inputs are two Excel sheets containing films and televisions' attributes. The headings of the form look like this:



	A	B	C	D	E	F	G	H	I	J	K	L
1	assets_id	assets_nam	show_type	year	director	actor	writer	region	country	summary	douban_lin	imdb

The name and the summary are the hardest assets to compare because they are in sentence/paragraph form. Hence, Gensim and `jieba` are used here.

After using `pandas` to extract all the data from the two Excel sheets and saving them in data frames, we can store the name and the summary separately in dictionaries. This helps indicate the ID of the assets and their corresponding attributes.

```
# This is the first excel file
ccms_title = {}
for i in range(len(df[‘assets_id’])):
 name = df[‘assets_name’][i]
 iD = str(df[‘assets_id’][i])
 ccms_title[iD] = name

# This is the second excel file
douban_title = {}
for i in range(len(df2[‘assets_id’])):
 name = df2[‘assets_name’][i]
 iD = str(df2[‘assets_id’][i])
 douban_title[iD] = name

ccms_summary = {}
for i in range(len(df['assets_id'])):
    name = df['assets_name'][i]
    iD = str(df['summary'][i])
    ccms_summary[iD] = name

douban_summary = {}
for i in range(len(df2['assets_id'])):
    name = df2['assets_name'][i]
    iD = str(df2['summary'][i])
    douban_summary[iD] = name
```

Create a function that uses Gensim to calculate the similarity score between titles and summary.

```
def gensimCalculation(d1, d2):
  new_dict = {}

  for x in d1:
    text1 = d1[x]
    texts = [jieba.lcut(d2[y]) for y in d2]
    dictionary = corpora.Dictionary(texts)
    feature_cnt = len(dictionary.token2id)
    corpus = [dictionary.doc2bow(text) for text in texts]
    tfidf = models.TfidfModel(corpus)
 print(new_dict)
 return

print(gensimCalculation(ccms_summary, douban_summary))
```

The function compares the summary of a single asset in excel1 with every other summary in excel2, and it finds the most similar summary. The output is a dictionary with the ID of the assets and the most similar asset ID with its most similar summary.

Other similarity calculations, such as comparing titles, can be efficiently generated in this way too.

Summary

When approaching Gensim, I learned to focus more on the input and the output in each step. With the help of `jieba`, the word segmentation module in Python, text similarity is easily calculated.