Genism

2201210498 尹雯婧

1. genism安装

```
pip install genism

Successfully installed Cython-0.29.28 gensim-4.2.0 smart-open-6.2.0
```

2. 使用方法

(1) 准备训练语料

假设列表 documents 代表语料库，每一句话代表一个文档，documents 中有9个元素，也就是说该语料库由9个文档组成。

```
from gensim import corpora

documents = ["Human machine interface for lab abc computer applications",
             "A survey of user opinion of computer system response time",
             "The EPS user interface management system",
             "System and human system engineering testing of EPS",
             "Relation of user perceived response time to error measurement",
             "The generation of random binary unordered trees",
             "The intersection graph of paths in trees",
             "Graph minors IV Widths of trees and well quasi ordering",
             "Graph minors A survey"]
```

(2) 预处理

分词（tokenize）、去除停用词（stopwords）和在语料中只出现一次的词。 处理语料的方式有很多，这里只通过空格（whitespace）去分词，把每个词变为小写，最后去除一些常用的词和只出现一次的词。

```
stoplist = set('for a of the and to in'.split())
texts = [[word for word in document.lower().split() if word not in stoplist]
for document in documents]
from collections import defaultdict
frequency = defaultdict(int)
for text in texts:
    for token in text:
        frequency[token] += 1
texts = [[token for token in text if frequency[token] > 1]
              for text in texts]

from pprint import pprint
pprint(texts)
```

输出结果为：

```
[['human', 'interface', 'computer'],
 ['survey', 'user', 'computer', 'system', 'response', 'time'],
 ['eps', 'user', 'interface', 'system'],
 ['system', 'human', 'system', 'eps'],
 ['user', 'response', 'time'],
 ['trees'],
 ['graph', 'trees'],
 ['graph', 'minors', 'trees'],
 ['graph', 'minors', 'survey']]
```

(3) 文本向量化

这里使用词袋模型（bag-of-words）来提取文档特征。该模型通过计算每个词在文档中出现的频率，然后将这些频率组成一个向量，从而将文档向量化。 首先我们需要用语料库训练一个词典，词典包含所有在语料库中出现的单词。

```
dictionary = corpora.Dictionary(texts)
dictionary.save('./deerwester.dict')
print(dictionary)
print(dictionary.token2id)
```

输出结果为：

```
Dictionary<12 unique tokens: ['computer', 'human', 'interface', 'response', 's
urvey']...>

{'computer': 0, 'human': 1, 'interface': 2, 'response': 3, 'survey': 4, 'syste
m': 5, 'time': 6, 'user': 7, 'eps': 8, 'trees': 9, 'graph': 10, 'minors': 11}
```

上面已经构建了单词词典，我们可以通过该词典用词袋模型将其他的文本向量化。

```
new_doc = "Human computer interaction"
new_vec = dictionary.doc2bow(new_doc.lower().split())
print(new_vec)
```

输出结果为：

输出结果为：

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

假设新文本是"Human computer interaction"，则输出向量为[(0, 1), (1, 1)]。 其中，(0, 1)中的"0"表示 computer 在词典中的 id 为 0，"1"表示 Human 在该文档中出现了1次。 同理，(1, 1)表示 Human 在词典中的 id 为 1，出现次数为 1。 interaction 不在词典中，所以没有任何对应输出。 通过以下操作，我们能够看到我们这次得到的语料库：

```
corpus = [dictionary.doc2bow(text) for text in texts]
corpora.MmCorpus.serialize('/out/deerwester.mm', corpus) # 存入硬盘，以备后需
print(corpus)
```

输出结果为：

```
[(0, 1), (1, 1), (2, 1)]
[(0, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]
[(2, 1), (5, 1), (7, 1), (8, 1)]
[(1, 1), (5, 2), (8, 1)]
[(3, 1), (6, 1), (7, 1)]
[(9, 1)]
[(9, 1), (10, 1)]
[(9, 1), (10, 1), (11, 1)]
[(4, 1), (10, 1), (11, 1)]
```

(4) 语料库流

在以上的训练过程中，一整个语料库作为一个 Python List 存在了内存中。如果语料库很大，这样的存储方式对内存很不友好。 我们可以一次取出一个文档，这样同一时间只有一个文档在内存中。

```
class MyCorpus(object):
    def __iter__(self):
        for line in open('mycorpus.txt'):
            yield dictionary.doc2bow(line.lower().split())
corpus_memory_friendly = MyCorpus()# 没有将corpus加载到内存中
print(corpus_memory_friendly)

for vector in corpus_memory_friendly:  # 每次加载一个向量放入内存
    print(vector)
```

输出结果为：

```
<__main__.MyCorpus object at 0x000001F4F23FDE50>
```

```
[(0, 1), (1, 1), (2, 1)]
[(0, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]
[(2, 1), (5, 1), (7, 1), (8, 1)]
[(1, 1), (5, 2), (8, 1)]
[(3, 1), (6, 1), (7, 1)]
[(9, 1)]
[(9, 1), (10, 1)]
[(9, 1), (10, 1), (11, 1)]
[(4, 1), (10, 1), (11, 1)]
```

相似地，为了构造 dictionary 我们也不必将全部文档读入内存：

```
from gensim import corpora
from six import iteritems
stoplist = set('for a of the and to in'.split())
#初步构建所有单词的词典
dictionary = corpora.Dictionary(line.lower().split() for line in open('mycorp
us.txt') )
#去出停用词,stop_ids表示停用词在dictionary中的id
stop_ids = [dictionary.token2id[stopword] for stopword in stoplist if stopwor
d in dictionary.token2id]
#只出现一次的单词id
once_ids = [tokenid for tokenid, docfreq in iteritems(dictionary.dfs) if docf
req ==1]
#根据stop_ids与once_ids清洗dictionary
dictionary.filter_tokens(stop_ids + once_ids)
# 去除清洗后的空位
dictionary.compactify()
print(dictionary)
```

输出结果为：

```
Dictionary<12 unique tokens: ['computer', 'human', 'interface', 'response', 's
urvey']...>
```

(5) 主题向量的变换

Genism的核心之一就是对文本向量的转换。通过挖掘语料中隐藏的语义结构特征，我们最终可以变换出一个简洁高效的文本向量。 在Gensim中，每一个向量变换的操作都对应着一个模型，例如上面提到的对应着词袋模型的doc2bow变换。 每一个模型又都是一个标准的Python对象。 下面以TF-IDF模型为例，介绍Gensim模型的一般使用方法。

首先是模型对象的初始化。通常，Gensim 模型接受一段训练语料作为初始化的参数。显然，越复杂的模型需要配置的参数越多。

```
from gensim import models
tfidf = models.TfidfModel(corpus)
```

其中，corpus是一个返回bow向量的迭代器。这两行代码将完成对corpus中出现的每一个特征的IDF值的统计工作。

接下来，我们可以调用这个模型将任意一段语料转化成TFIDF向量。 需要注意的是，这里的bow向量必须与训练语料的bow向量共享同一个特征字典（即共享同一个向量空间）。

```
doc_bow = [(0, 1), (1, 1)]
print(tfidf[doc_bow])
```

输出结果为：

```
[(0, 0.7071067811865476), (1, 0.7071067811865476)]
```

Gensim内置了多种主题模型的向量变换，包括LDA，LSI，RP，HDP等。 这些模型通常以bow向量或tfidf向量的语料为输入，生成相应的主题向量。所有的模型都支持流式计算。

(6) 文档相似度计算

在得到每一篇文档对应的主题向量后，我们就可以计算文档之间的相似度，进而完成如文本聚类、信息检索之类的任务。Gensim提供了这一类任务的API接口。

以信息检索为例。对于一篇待检索的query，我们的目标是从文本集合中检索出主题相似度最高的文档。

首先，我们需要将待检索的query和文本放在同一个向量空间里进行表达：

```
# 构造LSI模型并将待检索的query和文本转化为LSI主题向量
# 转换之前的corpus和query均是BOW向量
lsi_model = models.LsiModel(corpus, id2word=dictionary, num_topics=2)
documents = lsi_model[corpus]
query_vec = lsi_model[query]
```

接下来，我们用待检索的文档向量初始化一个相似度计算的对象：

```
index = similarities.MatrixSimilarity(documents)
```

最后，我们借助index对象计算任意一段query和所有文档的相似度：

```
sims = index[query_vec]
```

参考资料

Gensim: Topic modelling for humans (https://radimrehurek.com/gensim/)
Gensim 中文文档 (https://gensim.apachecn.org/#/)
Genism 入门教程 (https://www.cnblogs.com/iloveai/p/gensim_tutorial.html)