Gensim调研报告

任务要求

本仓库对于Gensim的分析基于翻译技术与实践课程中的作业，作业的要求如下：

基础任务：知道它是啥，能做什么

进阶任务：了解其所有可能的用途，跑通部分测试用例

Gensim介绍

Gensim是一款开源的第三方Python工具包，用于从原始的非结构化的文本中，无监督地学习到文本隐层的主题向量表达。

它支持包括TF-IDF，LSA，LDA，和word2vec在内的多种无监督的主题模型算法，支持流式训练，并提供了诸如相似度计算，信息检索等一些常用任务的API接口

本项目对gensim库的各个功能进行介绍和简单的使用

参考资料：

- https://www.geeksforgeeks.org/nlp-gensim-tutorial-complete-guide-for-beginners/
- Gensim中文文档 https://gensim.apachecn.org/#/
- Gensim官方文档 https://radimrehurek.com/gensim/auto_examples/index.html#documentation
- 15分钟入门Gensim https://zhuanlan.zhihu.com/p/37175253

Gensim功能

Gensim的主要功能有：

- **内存独立性**：任何时候都不需要整个训练语料库完全存放在内存中，因此可以处理大型的Web级语料库
- **内存共享**：经过训练的模型可以持久保存并通过mmap加载，多个进程可以共享数据从而减少内存占用
- **向量空间算法的高效实现**：包括Word2Vec、Doc2Vec、FastText、Tf-Idf、潜在语义分析（LSI，LSA）、隐含狄利克雷分布（LDA）和随机投影（RP）

- 对主流数据格式的I/O包装和读写
- 通过语义表示对文档进行相似性查询

Gensim的核心概念

语料Corpus

文本文档的集合

Corpus在Gensim中有两种作用

1. 作为模型训练的输入，模型通过训练语料查找文档的主题，初始化模型的参数
2. 根据训练后的主题模型从文档中提取主题，通过语义相似性、聚类等方式索引语料库

向量Vector

用数学方式表达文档，将文档转换为一串数字

比如：可以用词袋模型将文档表示为向量，每个向量包含字典中每个单词在该文档中的频率

通常情况下，向量中包含很多零值，为了减少内存占用，Gensim可以省略向量中所有值为0的元素，用稀疏向量（Sparse Vector）的格式表示文档。
如：(0.0, 2.0, 5.0) -> [(2, 2.0), (3, 5.0)]

模型 Model

模型指将文档从一种表示形式转换到另一种文档表示。在Gensim中文档通过向量表示，模型可以被认为是将向量从一种表示形式转换为另一种表示形式的算法

Gensim实现了Word2Vec、LsiModel、LdaModel等模型

Gensim功能展示

语料库和字典

Gensim中没有对于对原始文本进行预处理的方式和接口进行强制约束，可以使用不同的方法对原始文本（根据任务的需求）进行预处理：分词、处理停用词、词干提取、大小写转换等等。

Gensim本身提供了simple_preprocess方法进行预处理，方法会对文本进行分词和标准化之后返回token序列

In [1]:

```
import gensim
from gensim.utils import simple_preprocess

# open the text file as an object
doc = ["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"]

# preprocess the file to get a list of tokens
tokenized =[]
for sentence in doc:
  # the simple_preprocess function returns a list of each sentence
  tokenized.append(simple_preprocess(sentence, deacc = True))

print(tokenized)
```

```
[['human', 'machine', 'interface', 'for', 'lab', 'abc', 'computer', 'applications'], ['survey', 'o
f', 'user', 'opinion', 'of', 'computer', 'system', 'response', 'time'], ['the', 'eps', 'user', 'inte
rface', 'management', 'system'], ['system', 'and', 'human', 'system', 'engineering', 'testing', 'o
f', 'eps'], ['relation', 'of', 'user', 'perceived', 'response', 'time', 'to', 'error', 'measuremen
t'], ['the', 'generation', 'of', 'random', 'binary', 'unordered', 'trees'], ['the', 'intersection',
'graph', 'of', 'paths', 'in', 'trees'], ['graph', 'minors', 'iv', 'widths', 'of', 'trees', 'and', 'w
ell', 'quasi', 'ordering'], ['graph', 'minors', 'survey']]
```

对于经过预处理的文档，Gensim可以通过Dictionary类根据语料构建字典，字典对语料中每个单词赋予一个整数索引，根据这个索引可以使用Dictionary的方法将句子转换为词袋模型的向量

```
from gensim import corpora
my_dictionary = corpora.Dictionary(tokenized)
print(my_dictionary)
print(my_dictionary.token2id) # token到其id的map
print(my_dictionary.doc2bow(tokenized[0])) # 将文本转换为词袋模型的向量
```

```
Dictionary<41 unique tokens: ['abc', 'applications', 'computer', 'for', 'human']...>
{'abc': 0, 'applications': 1, 'computer': 2, 'for': 3, 'human': 4, 'interface': 5, 'lab': 6, 'machin
e': 7, 'of': 8, 'opinion': 9, 'response': 10, 'survey': 11, 'system': 12, 'time': 13, 'user': 14, 'e
ps': 15, 'management': 16, 'the': 17, 'and': 18, 'engineering': 19, 'testing': 20, 'error': 21, 'mea
surement': 22, 'perceived': 23, 'relation': 24, 'to': 25, 'binary': 26, 'generation': 27, 'random':
28, 'trees': 29, 'unordered': 30, 'graph': 31, 'in': 32, 'intersection': 33, 'paths': 34, 'iv': 35,
'minors': 36, 'ordering': 37, 'quasi': 38, 'well': 39, 'widths': 40}
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]
```

Gensim提供了不同的方法存储和读取字典，支持通过Matrix Market格式保存和读取语料库

```
# 通过Dictionary默认存储方式 保存和打开字典
my_dictionary.save('my_dict.dict')
load_dict = corpora.Dictionary.load('my_dict.dict')
print(load_dict.token2id) # token到其id的map
print(load_dict.doc2bow(tokenized[0])) # 将文本转换为词袋模型的向量

# 将字典用文本文件读取
from gensim.test.utils import get_tmpfile
tmp_fname = get_tmpfile('dictionary')
my_dictionary.save_as_text(tmp_fname)
load_dict = corpora.Dictionary.load_from_text(tmp_fname)
print(load_dict.token2id) # token到其id的map
print(load_dict.doc2bow(tokenized[0])) # 将文本转换为词袋模型的向量
```

```
{'abc': 0, 'applications': 1, 'computer': 2, 'for': 3, 'human': 4, 'interface': 5, 'lab': 6, 'machin
e': 7, 'of': 8, 'opinion': 9, 'response': 10, 'survey': 11, 'system': 12, 'time': 13, 'user': 14, 'e
ps': 15, 'management': 16, 'the': 17, 'and': 18, 'engineering': 19, 'testing': 20, 'error': 21, 'mea
surement': 22, 'perceived': 23, 'relation': 24, 'to': 25, 'binary': 26, 'generation': 27, 'random':
28, 'trees': 29, 'unordered': 30, 'graph': 31, 'in': 32, 'intersection': 33, 'paths': 34, 'iv': 35,
'minors': 36, 'ordering': 37, 'quasi': 38, 'well': 39, 'widths': 40}
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]
{'abc': 0, 'and': 18, 'applications': 1, 'binary': 26, 'computer': 2, 'engineering': 19, 'eps': 15,
'error': 21, 'for': 3, 'generation': 27, 'graph': 31, 'human': 4, 'in': 32, 'interface': 5, 'interse
ction': 33, 'iv': 35, 'lab': 6, 'machine': 7, 'management': 16, 'measurement': 22, 'minors': 36, 'o
f': 8, 'opinion': 9, 'ordering': 37, 'paths': 34, 'perceived': 23, 'quasi': 38, 'random': 28, 'relat
ion': 24, 'response': 10, 'survey': 11, 'system': 12, 'testing': 20, 'the': 17, 'time': 13, 'to': 2
5, 'trees': 29, 'unordered': 30, 'user': 14, 'well': 39, 'widths': 40}
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]
```

```
from gensim.corpora import MmCorpus
from gensim.test.utils import get_tmpfile

BoW_corpus =[my_dictionary.doc2bow(doc, allow_update = True) for doc in tokenized]
print(BoW_corpus) # 将文本转换为词袋模型的向量

output_fname = get_tmpfile("BoW_corpus.mm")
# 存储语料库
MmCorpus.serialize(output_fname, BoW_corpus)
# 读取语料库
load_corpus = MmCorpus(output_fname)
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)], [(2, 1), (8, 2), (9, 1), (10, 1),
(11, 1), (12, 1), (13, 1), (14, 1)], [(5, 1), (12, 1), (14, 1), (15, 1), (16, 1), (17, 1)], [(4, 1),
(8, 1), (12, 2), (15, 1), (18, 1), (19, 1), (20, 1)], [(8, 1), (10, 1), (13, 1), (14, 1), (21, 1),
(22, 1), (23, 1), (24, 1), (25, 1)], [(8, 1), (17, 1), (26, 1), (27, 1), (28, 1), (29, 1), (30, 1)],
[(8, 1), (17, 1), (29, 1), (31, 1), (32, 1), (33, 1), (34, 1)], [(8, 1), (18, 1), (29, 1), (31, 1),
(35, 1), (36, 1), (37, 1), (38, 1), (39, 1), (40, 1)], [(11, 1), (31, 1), (36, 1)]]
```

Gensim中使用TF-IDF模型

TF-IDF — Term Frequency(词频)-Inverse Document Frequency(逆文档频率)，是一种用于信息检索和数据挖掘的加权技术。

字词的重要性与它在文件中出现的次数成正比增加，但同时会随着在语料库中出现的频率成反比下降。

TF-IDF的主要思想是，如果某个词在文档中出现的频率高，在其他文档中很少出现，认为这个词具有很好的类别区分能力，适合用来分类。

- 优点：实现简单高效
- 缺点：不能有效反映词的重要程度，没有考虑到特征词的分布情况(特征词的位置，在不同类别文档中的分布)，

参考资料：https://blog.csdn.net/asialee_bird/article/details/81486700

```
from gensim import models
import numpy as np

# 计算词袋模型中的token权重
word_weight =[]
for doc in BoW_corpus:
    for id, freq in doc:
        word_weight.append([my_dictionary[id], freq])
print(word_weight)
```

```
[['abc', 1], ['applications', 1], ['computer', 1], ['for', 1], ['human', 1], ['interface', 1], ['la
b', 1], ['machine', 1], ['computer', 1], ['of', 2], ['opinion', 1], ['response', 1], ['survey', 1],
['system', 1], ['time', 1], ['user', 1], ['interface', 1], ['system', 1], ['user', 1], ['eps', 1],
['management', 1], ['the', 1], ['human', 1], ['of', 1], ['system', 2], ['eps', 1], ['and', 1], ['eng
ineering', 1], ['testing', 1], ['of', 1], ['response', 1], ['time', 1], ['user', 1], ['error', 1],
['measurement', 1], ['perceived', 1], ['relation', 1], ['to', 1], ['of', 1], ['the', 1], ['binary',
1], ['generation', 1], ['random', 1], ['trees', 1], ['unordered', 1], ['of', 1], ['the', 1], ['tree
s', 1], ['graph', 1], ['in', 1], ['intersection', 1], ['paths', 1], ['of', 1], ['and', 1], ['trees',
1], ['graph', 1], ['iv', 1], ['minors', 1], ['ordering', 1], ['quasi', 1], ['well', 1], ['widths',
1], ['survey', 1], ['graph', 1], ['minors', 1]]
```

```
tfIdf = models.TfidfModel(BoW_corpus, smartirs = 'ntc')

# 查看tf-idf权重
weight_tfidf =[]
for doc in tfIdf[BoW_corpus]:
    for id, freq in doc:
        weight_tfidf.append([my_dictionary[id], np.around(freq, decimals = 3)])
print(weight_tfidf)
```

```
[['abc', 0.393], ['applications', 0.393], ['computer', 0.275], ['for', 0.393], ['human', 0.275], ['i
nterface', 0.275], ['lab', 0.393], ['machine', 0.393], ['computer', 0.363], ['of', 0.231], ['opinio
n', 0.52], ['response', 0.363], ['survey', 0.363], ['system', 0.272], ['time', 0.363], ['user', 0.27
2], ['interface', 0.418], ['system', 0.313], ['user', 0.313], ['eps', 0.418], ['management', 0.598],
['the', 0.313], ['human', 0.326], ['of', 0.103], ['system', 0.487], ['eps', 0.326], ['and', 0.326],
['engineering', 0.466], ['testing', 0.466], ['of', 0.088], ['response', 0.278], ['time', 0.278], ['u
ser', 0.208], ['error', 0.398], ['measurement', 0.398], ['perceived', 0.398], ['relation', 0.398],
['to', 0.398], ['of', 0.103], ['the', 0.244], ['binary', 0.466], ['generation', 0.466], ['random',
0.466], ['trees', 0.244], ['unordered', 0.466], ['of', 0.113], ['the', 0.266], ['trees', 0.266], ['g
raph', 0.266], ['in', 0.508], ['intersection', 0.508], ['paths', 0.508], ['of', 0.087], ['and', 0.27
3], ['trees', 0.204], ['graph', 0.204], ['iv', 0.39], ['minors', 0.273], ['ordering', 0.39], ['quas
i', 0.39], ['well', 0.39], ['widths', 0.39], ['survey', 0.625], ['graph', 0.468], ['minors', 0.625]]
```

Gensim中N-gram模型使用

Gensim中支持N-gram模型，通过gensim.models.phrases模块

```
from gensim.models.phrases import Phrases

data = tokenized
# 使用Phraser模型构建Bigram模型
bigram_model = Phrases(data, min_count = 1, threshold = 2)
print(bigram_model[data[0]])

# 使用Phraser模型构建Trigram模型
trigram_model = Phrases(bigram_model[data], threshold = 2)
print(trigram_model[bigram_model[data[0]]])
```

```
['human', 'machine', 'interface', 'for', 'lab', 'abc', 'computer', 'applications']
['human', 'machine', 'interface', 'for', 'lab', 'abc', 'computer', 'applications']
```

Gensim中Word2Vec模型使用

Word2Vec是一种Word Embedding方法，将不可计算、非结构化的词转化为可计算、结构化的向量，即将词转换为词向量。

Word2Vec模型基于词袋模型和Skip-gram模型，使用双层神经网络用来训练来重新构建词向量。

参考资料：https://zhuanlan.zhihu.com/p/26306795

```
from multiprocessing import cpu_count
from gensim.models.word2vec import Word2Vec

# 将数据集划分为两部分，一部分训练模型，另一部分更新模型
data_1 = data[:5]
data_2 = data[5:]

# 训练Word2Vec模型
w2v_model = Word2Vec(data_1, min_count = 0, workers = cpu_count())

# 展示Word2Vec中的词向量
print(w2v_model.wv['of'])
```

```
[-8.6178398e-03  3.6644570e-03  5.1917033e-03  5.7428684e-03
  7.4663050e-03 -6.1683957e-03  1.1092245e-03  6.0532028e-03
 -2.8456689e-03 -6.1792773e-03 -4.0824802e-04 -8.3722677e-03
 -5.5993502e-03  7.1130781e-03  3.3511671e-03  7.2302753e-03
  6.8049147e-03  7.5388737e-03 -3.7864440e-03 -5.7406613e-04
  2.3495727e-03 -4.5223944e-03  8.3973231e-03 -9.8532550e-03
  6.7634289e-03  2.9218255e-03 -4.9348911e-03  4.3953261e-03
 -1.7408995e-03  6.7140339e-03  9.9721309e-03 -4.3585496e-03
 -5.9606915e-04 -5.6982320e-03  3.8543681e-03  2.7912222e-03
```

Word2Vec将词转换为词向量（包含其语义表示），可以通过向量的相似度表示词之间的相似度

```
print(w2v_model.wv.most_similar('of'))
```

```
[('error', 0.18891339004039764), ('lab', 0.16071486473083496), ('the', 0.15929096937179565), ('for', 0.13740161061286926), ('engineering', 0.12791527807712555), ('abc', 0.12299858033657074), ('survey', 0.08541512489318848), ('eps', 0.06807880848646164), ('human', 0.03370113670825958), ('perceived', 0.022662857547402382)]
```

Gensim中Doc2Vec的使用

Doc2vec方法是一种无监督算法，能从变长的文本（例如：句子、段落或文档）中学习得到固定长度的特征表示，用一个向量表示不同的文档，潜在地克服词袋模型的缺点。

参考资料：https://zhuanlan.zhihu.com/p/136096645

```
from gensim.models import doc2vec

# To train the model we need a list of tagged documents
def tagged_document(list_of_ListOfWords):
    for x, ListOfWords in enumerate(list_of_ListOfWords):
        yield doc2vec.TaggedDocument(ListOfWords, [x])

# 训练数据
data_train = list(tagged_document(data))
print(data_train[:1])
```

```
[TaggedDocument(words=['human', 'machine', 'interface', 'for', 'lab', 'abc', 'computer', 'applications'], tags=[0])]
```

```
d2v_model = doc2vec.Doc2Vec(vector_size = 40, min_count = 2, epochs = 30)
d2v_model.build_vocab(data_train)
d2v_model.train(data_train, total_examples = d2v_model.corpus_count, epochs = d2v_model.epochs)
Analyze = d2v_model.infer_vector(['graph', 'is', 'for', 'human']) # 训练好的模型可以得出一个文本的向
print(Analyze)
```

```
[ 0.00700085 -0.00449182  0.00046063 -0.00601998  0.00345083  0.00598205
  0.00239306  0.00987917  0.00242983 -0.01217873 -0.0125926   0.00911747
 -0.00233731 -0.00531576  0.00329072 -0.01189259 -0.00873746  0.0076514
  0.00873276  0.00451682  0.00431774  0.00380691 -0.00602418  0.00167289
  0.00761709 -0.00901803  0.0011268   0.01134675 -0.00301049  0.00633963
 -0.01125128  0.00887871  0.0074078  -0.00197911 -0.01018545 -0.00181363
 -0.01065868  0.00099068  0.00870525 -0.00550304]
```

Gensim中LDA的使用

LDA - Latent Dirichlet Allocation，用来推测文档的主题分布。它可以将文档集中每篇文档的主题以概率分布的形式给出，从而通过分析一些文档抽取出它们的主题分布后，便可以根据主题分布进行主题聚类或文本分类。

LDA是一种无监督学习，基于词袋模型，它认为文档是一组词构成的集合，词与词之间是无序的。一篇文档可以包含多个主题，文档中的每个词都是由某个主题生成的，LDA给出文档属于每个主题的概率分布，同时给出每个主题上词的概率分布。

参考资料：

- https://zhuanlan.zhihu.com/p/31470216
- https://blog.csdn.net/guleileo/article/details/80971601

```
from gensim.models import LdaModel, LdaMulticore
# 训练模型
lda_model = LdaModel(corpus = BoW_corpus, num_topics = 2)
# 保存模型
lda_model.save('LDA_model.model')
# 展示提取出的主题
print(lda_model.print_topics(-1))
```

```
[(0, '0.059*"8" + 0.043*"12" + 0.039*"5" + 0.038*"2" + 0.036*"4" + 0.035*"14" + 0.033*"17" + 0.030
*"13" + 0.030*"15" + 0.029*"1"'), (1, '0.086*"8" + 0.049*"12" + 0.046*"31" + 0.043*"29" + 0.040*"17"
+ 0.039*"14" + 0.037*"36" + 0.035*"18" + 0.034*"11" + 0.029*"10"')]
```

LDA模型主要提供3个方面的信息:

- 文档中的主题
- 每个词属于什么主题
- φ值

```
# 某个词属于某个主题的概率
print(lda_model.get_term_topics(my_dictionary.token2id['human']))

# 将词转换为词袋模型的序列
bow_list =['graph', 'path', 'tree']
bow = my_dictionary.doc2bow(bow_list)

# 展示文本的主题，单词的主题，可能性
doc_topics, word_topics, phi_values = lda_model.get_document_topics(bow, per_word_topics = True)
print(doc_topics)
print(word_topics)
print(phi_values)
```

```
[(0, 0.026856301), (1, 0.013624351)]
[(0, 0.28911325), (1, 0.7108867)]
[(31, [1, 0])]
[(31, [(0, 0.07797001), (1, 0.92202544)])]
```

Gensim中使用LSI模型

隐性语义索引 LSI — Latent Semantic Indexing，是一种简单实用的主题模型。它是一种利用奇异值分解(SVD)方法获得在文本中术语和概念之间关系的索引和获取方法。该方法的主要依据是在相同文章中的词语一般有类似的含义,可以从一篇文章中提取术语关系，从而建立起主要概念内容。

参考资料

- https://www.cnblogs.com/pinard/p/6805861.html
- https://blog.csdn.net/guoziqing506/article/details/81007364

```
from gensim.models import LsiModel

lsi_model = LsiModel(corpus = BoW_corpus, id2word = my_dictionary, num_topics = 2, decay = 0.5)
# 展示提取的主题
print(lsi_model.print_topics(-1))
```

```
[(0, '0.622*"of" + 0.359*"system" + 0.256*"user" + 0.206*"time" + 0.206*"response" + 0.197*"trees" +
0.170*"the" + 0.168*"and" + 0.158*"graph" + 0.147*"computer"'), (1, '0.399*"trees" + 0.364*"graph" +
-0.295*"system" + 0.245*"minors" + -0.226*"user" + 0.194*"widths" + 0.194*"quasi" + 0.194*"well" +
0.194*"ordering" + 0.194*"iv"')]
```

Gensim中相似性度量

将词或者文档转换为向量表示后，可以通过向量度量其相似性,将文档放在模型中计算其文档向量，之后使用相似度计算对象计算相似度。

Gensim中主要有两种度量方式

- 余弦相似度：用两个非零向量的夹角的余弦值度量相似度
- 软余弦相似度：与余弦相似度类似，但是软余弦相似度会考虑到向量空间模型中特征的相似度

默认使用余弦相似度

```
# 直接查看文本的主题
doc = "Human computer interaction".lower().split()
vec_bow = my_dictionary.doc2bow(doc)
vec_lsi = lsi_model[vec_bow]
print(vec_lsi)
```

```
[(0, 0.25908928880377124), (1, -0.2971336854885933)]
```

```
from gensim import similarities

index = similarities.MatrixSimilarity(lsi_model[BoW_corpus])
sims = index[vec_lsi]
print(list(enumerate(sims)))
```

```
[(0, 0.97791314), (1, 0.8738079), (2, 0.96754515), (3, 0.89737785), (4, 0.9269992), (5, 0.050218582),
(6, -0.06976479), (7, -0.21118912), (8, -0.24361911)]
```