

从本专栏开始，作者正式开始研究Python深度学习、神经网络及人工智能相关知识。前一篇详细讲解了卷积神经网络CNN原理，并通过TensorFlow编写CNN实现了MNIST分类学习案例。本篇文章将分享gensim词向量Word2Vec安装、基础用法，并实现《庆余年》中文短文本相似度计算及多个案例。本专栏主要结合作者之前的博客、AI经验和相关文章及论文介绍，后面随着深入会讲解更多的Python人工智能案例及应用。

基础性文章，希望对您有所帮助，如果文章中存在错误或不足之处，还请海涵~作者作为人工智能的菜鸟，希望大家能与我在这一笔一划的博客中成长起来。写了这么多年博客，尝试第一个付费专栏，但更多博客尤其基础性文章，还是会继续免费分享，但该专栏也会用心撰写，望对得起读者，共勉！

范闲和林婉儿的相似度为： 0.7795434

和范闲最相关的词有：

他 0.8562162518501282
夏栖飞 0.8079931735992432
洪竹 0.7994407415390015
五竹 0.7934709787368774
言冰云 0.7929771542549133
婉儿 0.7903158068656921
林婉儿 0.7795432209968567
明青达 0.7663493752479553
陈萍萍 0.7574742436408997
她 0.7511939406394958

和五竹最相关的词有：

肖恩 0.7937331199645996
范闲 0.7934710383415222
他 0.7675401568412781
言冰云 0.7527633309364319
陈萍萍 0.7523936033248901
海棠 0.7275645732879639
洪竹 0.7250131964683533
她 0.7217494249343872
话题 0.7188960313796997
林婉儿 0.712841272354126

<https://blog.csdn.net/eastmount>

代码下载地址：<https://github.com/eastmountyxz/Al-for-TensorFlow>

文章目录

一.Word2Vec原理

- 1.统计语言模型
- 2.神经网络概率语言模型
- 3.词向量
- 4.Word2vec

二.Word2Vec安装及入门

- 1.安装
- 2.基础用法

三.Word2Vec+DBSCAN短文本聚类及可视化

四.Word2Vec计算《庆余年》中文短文本相似度

五.Word2Vec写诗详解

六.总结

同时推荐前面作者另外三个Python系列文章。从2014年开始，作者主要写了三个Python系列文章，分别是基础知识、网络爬虫和数据分析。2018年陆续增加了Python图像识别和Python人工智能专栏。

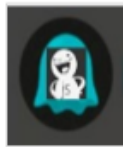
- Python基础知识系列：Python基础知识学习与提升
- Python网络爬虫系列：Python爬虫之Selenium+Phantomjs+CasperJS
- Python数据分析系列：知识图谱、web数据挖掘及NLP
- Python图像识别系列：Python图像处理及图像识别
- Python人工智能系列：Python人工智能及知识图谱实战



Python学习系列

文章：16篇

阅读：119908



Python爬虫之Selenium+Phantomjs+CasperJS

文章：33篇

阅读：443874



知识图谱、web数据挖掘及NLP

文章：44篇

阅读：488758

前文：

[Python人工智能] 一.TensorFlow2.0环境搭建及神经网络入门

[Python人工智能] 二.TensorFlow基础及一元直线预测案例

[Python人工智能] 三.TensorFlow基础之Session、变量、传入值和激励函数

[Python人工智能] 四.TensorFlow创建回归神经网络及Optimizer优化器

[Python人工智能] 五.Tensorboard可视化基本用法及绘制整个神经网络

[Python人工智能] 六.TensorFlow实现分类学习及MNIST手写体识别案例

[Python人工智能] 七.什么是过拟合及dropout解决神经网络中的过拟合问题

[Python人工智能] 八.卷积神经网络CNN原理详解及TensorFlow编写CNN

一.Word2Vec原理

作者第一次接触Word2Vec是2015年左右，那时候LDA+Word2Vec系列的论文都比较火，现在图书情报领域应用也挺多的，而自然语言处理领域BiLSTM+CNN+Attention相对流行。这个系列专栏不只是讲解TensorFlow知识，也会围绕NLP和AI前沿技术及论文、图像识别、语义识别、数据挖掘、恶意代码识别、情报分析等案例进行补充，本文就带领大家走进Word2Vec的世界，希望能帮助到大家！

参考前文：[word2vec词向量训练及中文文本相似度计算](#)

1.统计语言模型

统计语言模型的一般形式是给定已知的一组词，求解下一个词的条件概率。形式如下：

$$\hat{p}(w_1^T) = \prod_{i=1}^T \hat{p}(w_i | w_1^{i-1})$$

w_i 是第 i 个词, w_i^j 为上文词序列:
 $w_i^j = (w_i, w_{i+1}, w_{i+2}, \dots, w_{i-1}, w_i)$

统计语言模型的一般形式直观、准确, n-gram模型中假设在不改变词语在上下文中的顺序前提下, 距离相近的词语关系越近, 距离较远的关联度越远, 当距离足够远时, 词语之间则没有关联度。

但该模型没有完全利用语料的信息:

- (1) 没有考虑距离更远的词语与当前词的关系, 即超出范围n的词被忽略了, 而这两者很可能有关系的。

例如, “华盛顿是美国的首都”是当前语句, 隔了大于n个词的地方又出现了“北京是中国的首都”, 在n元模型中“华盛顿”和“北京”是没有关系的, 然而这两个句子却隐含了语法及语义关系, 即“华盛顿”和“北京”都是名词, 并且分别是美国和中国的首都。

- (2) 忽略了词语之间的相似性, 即上述模型无法考虑词语的语法关系。

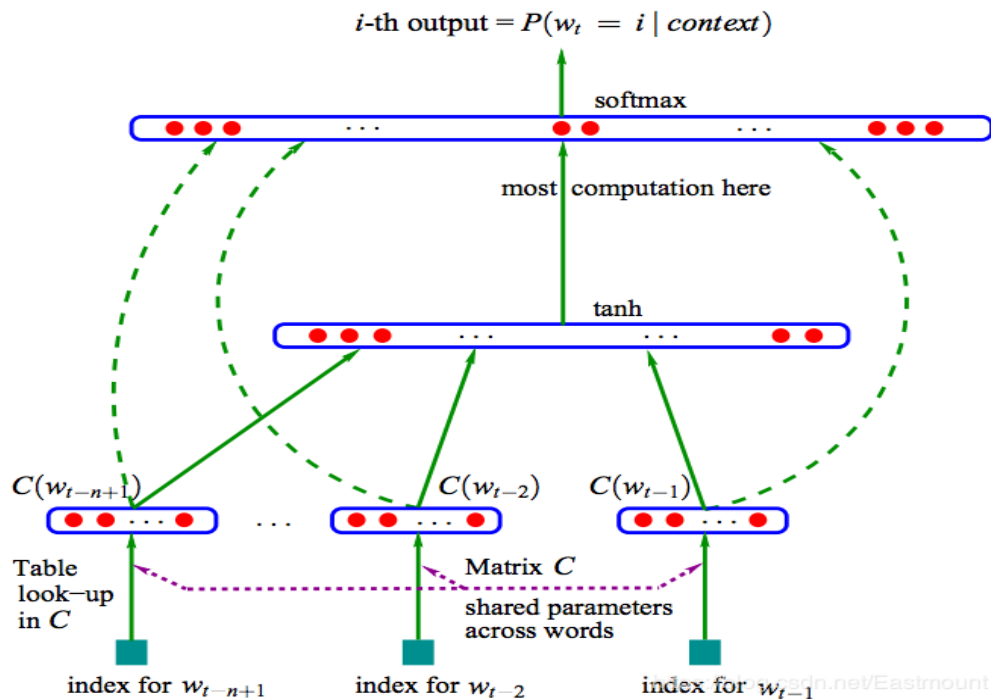
例如, 语料中的“鱼在水中游”应该能够帮助我们产生“马在草原上跑”这样的句子, 因为两个句子中“鱼”和“马”、“水”和“草原”、“游”和“跑”、“中”和“上”具有相同的语法特性。

而在神经网络概率语言模型中, 这两种信息将充分利用到。

2.神经网络概率语言模型

神经网络概率语言模型是一种新兴的自然语言处理算法, 该模型通过学习训练语料获取词向量和概率密度函数, 词向量是多维实数向量, 向量中包含了自然语言中的语义和语法关系, 词向量之间余弦距离的大小代表了词语之间关系的远近, 词向量的加减运算则是计算机在“遣词造句”。

神经网络概率语言模型经历了很长的发展阶段, 由Bengio等人2003年提出的神经网络语言模型NNLM (Neural network language model) 最为知名, 以后的发展工作都参照此模型进行。历经十余年的研究, 神经网络概率语言模型有了很大发展。如今在架构方面有比NNLM更简单的CBOW模型、Skip-gram模型; 其次在训练方面, 出现了Hierarchical Softmax算法、负采样算法 (Negative Sampling), 以及为了减小频繁词对结果准确性和训练速度的影响而引入的欠采样 (Subsampling) 技术。



上图是基于三层神经网络的自然语言估计模型NNLM(Neural Network Language Model)。NNLM可以计算某一个上下文的下一个词为 w_i 的概率，即 $P(w_i=i|context)$ ，词向量是其训练的副产物，NNLM根据语料库 C 生成对应的词汇表 V 。

近年来，神经网络概率语言模型发展迅速，Word2vec是最新技术理论的合集。Word2vec是Google公司在2013年开放的一款用于训练词向量的软件工具。所以，在讲述word2vec之前，先给大家介绍词向量的概念。

3.词向量

这里推荐licstar大神的NLP文章：Deep Learning in NLP （一）词向量和语言模型

正如作者所说：Deep Learning 算法已经在图像和音频领域取得了惊人的成果，但是在 NLP 领域中尚未见到如此激动人心的结果。有一种说法是，语言（词、句子、篇章等）属于人类认知过程中产生的高层认知抽象实体，而语音和图像属于较为底层的原始输入信号，所以后者更适合做deep learning来学习特征。

但是将词用“词向量”的方式表示可谓是将Deep Learning算法引入NLP领域的一个核心技术。自然语言理解问题转化为机器学习问题的第一步都是通过一种方法把这些符号数字化。

词向量具有良好的语义特性，是表示词语特征的常用方式。词向量的每一维的值代表一个具有一定的语义和语法上解释的特征。故可以将词向量的每一维称为一个词语特征。词向量用Distributed Representation表示，一种低维实数向量。

例如，NLP中最直观、最常用的词表示方法是One-hot Representation。每个词用一个很长的向量表示，向量的维度表示词表大小，绝大多数是0，只有一个维度是1，代表当前词。“话筒”表示为 $[0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \dots]$ 即从0开始话筒记为3。

但这种One-hot Representation采用稀疏矩阵的方式表示词，在解决某些任务时会造成维数灾难，而使用低维的词向量就很好的解决了该问题。同时从实践上看，高维的特征如果要套用Deep Learning，其复杂度几乎是难以接受的，因此低维的词向量在这里也饱受追捧。Distributed Representation低维实数向量，如：[0.792, -0.177, -0.107, 0.109, -0.542, ...]。它让相似或相关的词在距离上更加接近。

总之，Distributed Representation是一个稠密、低维的实数向量，它的每一维表示词语的一个潜在特征，该特征捕获了有用的句法和语义特征。其特点是将词语的不同句法和语义特征分布到它的每一个维度上去表示。

4.Word2vec

Word2vec是Google公司在2013年开放的一款用于训练词向量的软件工具。它根据给定的语料库，通过优化后的训练模型快速有效的将一个词语表达成向量形式，其核心架构包括CBOW和Skip-gram。

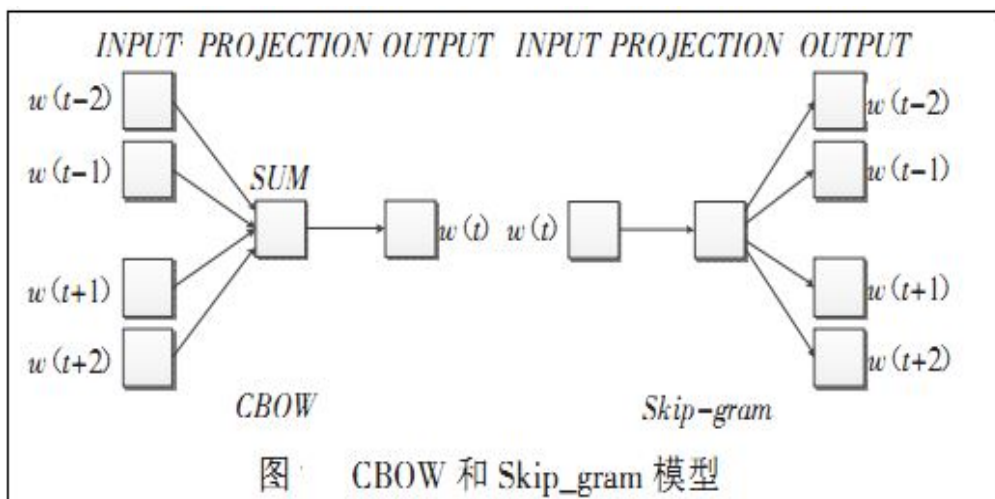
在开始之前，引入模型复杂度，定义如下：

$$O = E * T * Q$$

其中，E表示训练的次数，T表示训练语料中词的个数，Q因模型而异。E值不是我们关心的内容，T与训练语料有关，其值越大模型就越准确，Q在下面讲述具体模型是讨论。

NNLM模型是神经网络概率语言模型的基础模型。在NNLM模型中，从隐含层到输出层的计算是主要影响训练效率的地方，CBOW和Skip-gram模型考虑去掉隐含层。实践证明新训练的词向量的精确度可能不如NNLM模型（具有隐含层），但可以通过增加训练语料的方法来完善。

Word2vec包含两种训练模型，分别是CBOW和Skip-gram(输入层、发射层、输出层)，如下图所示：



CBOW模型：

理解为上下文决定当前词出现的概率。在CBOW模型中，上下文所有的词对当前词出现概率的影响的权重是一样的，因此叫CBOW(continuous bag-of-words model)模型。如在袋子中取词，取出数量足够的词就可以了，至于取出的先后顺序是无关紧要的。

Skip-gram模型：

Skip-gram模型是一个简单实用的模型。为什么会提出该模型呢？

在NLP中，语料的选取是一个相当重要的问题。

首先，语料必须充分。一方面词典的词量要足够大，另一方面尽可能地包含反映词语之间关系的句子，如“鱼在水中游”这种句式在语料中尽可能地多，模型才能学习到该句中的语义和语法关系，这和人类学习自然语言是一个道理，重复次数多了，也就学会模型了。

其次，语料必须准确。所选取的语料能够正确反映该语言的语义和语法关系，如中文的《人民日报》比较准确。但更多时候不是语料选取引发准确性问题，而是处理的方法。

由于窗口大小的限制，这会导致超出窗口的词语与当前词之间的关系不能正确地反映到模型中，如果单纯扩大窗口大小会增加训练的复杂度。Skip-gram模型的提出很好解决了这些问题。

Skip-gram表示“跳过某些符号”。例如句子“中国足球踢得真是太烂了”有4个3元词组，分别是“中国足球踢得”、“足球踢得真是”、“踢得真是太烂”、“真是太烂了”，句子的本意都是“中国足球太烂”，可是上面4个3元组并不能反映出这个信息。

此时，使用Skip-gram模型允许某些词被跳过，因此可组成“中国足球太烂”这个3元词组。如果允许跳过2个词，即2-Skip-gram，那么上句话组成的3元词组为：

Table 1 The example of Skip-gram	
Tri-grams	2-Skip-gram-tri-grams
“中国足球踢得”、“足球踢得真是”、“踢得真是太烂”、“真是太烂了”	“中国足球踢得”、“中国足球真是”、“中国足球太烂”、“中国踢得真是”、“中国踢得太烂”、“中国踢得了”、“中国真是太烂”、“中国真是了”、“足球踢得真是”、“足球踢得太烂”、“足球踢得了”、“足球真是太烂”、“足球真是了”、“足球太烂了”、“踢得真是太烂”、“踢得真是了”、“踢得太烂了”、“真是太烂了”

由上表可知：一方面Skip-gram反映了句子的真实意思，在新组成的这18个3元词组中，有8个词组能够正确反映例句中的真实意思；另一方面，扩大了语料，3元词组由原来的4个扩展到了18个。语料的扩展能够提高训练的准确度，获得的词向量更能反映真实的文本含义。

Word2Vec总结

Word2Vec是通过深度学习将词表征为数值型向量的工具。它把文本内容简化处理，把词当做特征，Word2Vec将特征映射到K维向量空间，为文本数据寻求更加深层次的特征表示。Word2Vec获得的词向量可被用于聚类、找同义词、词性分析等。通过词之间的距离（如cosine相似度、欧氏距离等）来判断它们之间的语义相似度，采用一个三层的神经网络“输入层-隐层-输出层”。Word2Vec有个核心的技术是根据词频用Huffman编码

，使得所有词频相似的词隐藏层激活的内容基本一致，出现频率越高的词语，他们激活的隐藏层数目越少，这样有效的降低了计算的复杂度。

Word2Vec优点如下：

- 高效、考虑词语上下文的语义关系
- 与潜在语义分析（Latent Semantic Index, LSI）、潜在狄立克雷分配（Latent Dirichlet Allocation, LDA）的经典过程相比，Word2vec利用了词的上下文，语义信息更加地丰富
- 在医疗项目（诊断报告）、短信舆情分析、社交网络评价分析等领域，短文本很常见，因此word2vec会达到很好的语义表征效果

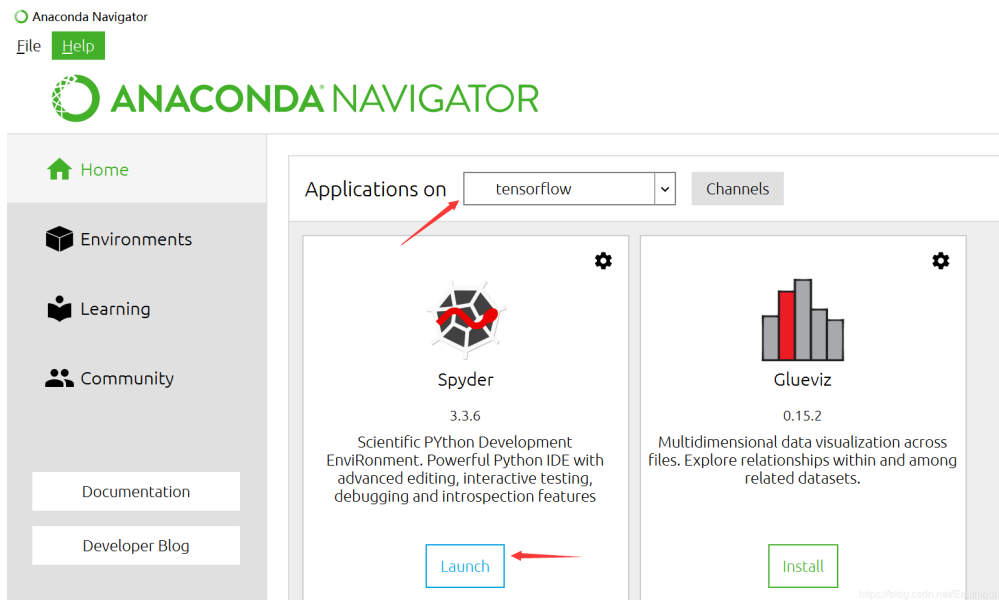
PS：这里推荐大家阅读熊富林等老师论文《Word2vec的核心架构及其应用》。

二.Word2Vec安装及入门

Word2Vec原理知识讲解完毕之后，我们开始尝试在Anaconda的TensorFlow环境中安装该扩展包。

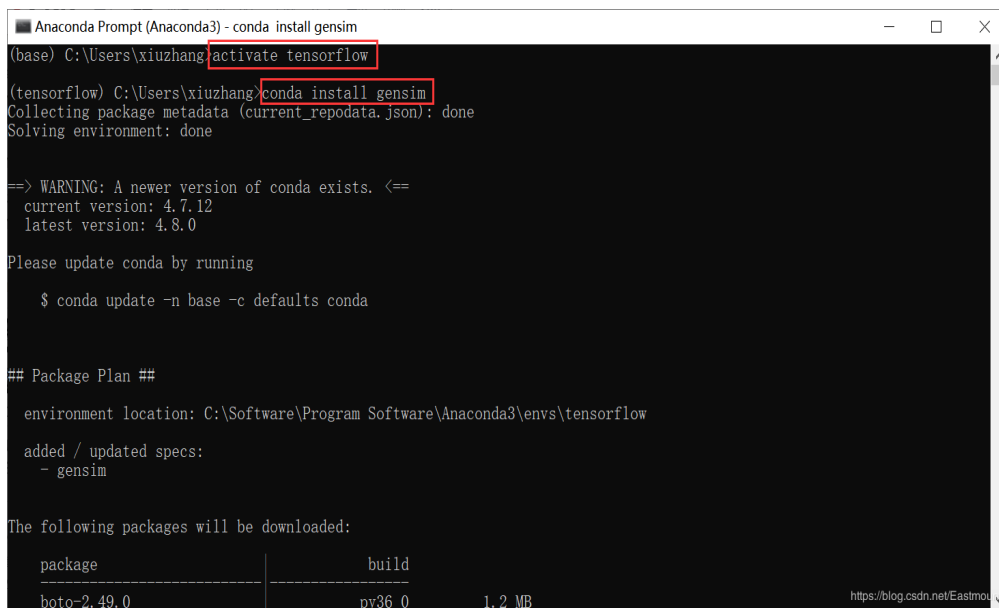
1.安装

第一步，打开Anaconda程序，并选择已经安装好的“TensorFlow”环境，运行Spyder。



第二步，我们需要在TensorFlow环境中安装gensim扩展包，否则会提示错误“ModuleNotFoundError: No module named ‘gensim’”。调用Anaconda Prompt安装即可，如下图所示：

```
activate tensorflow
conda install gensim
```



```

Anaconda Prompt (Anaconda3) - conda install gensim
(base) C:\Users\Xiuzhang>activate tensorflow
(tensorflow) C:\Users\Xiuzhang>conda install gensim
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
current version: 4.7.12
latest version: 4.8.0
Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

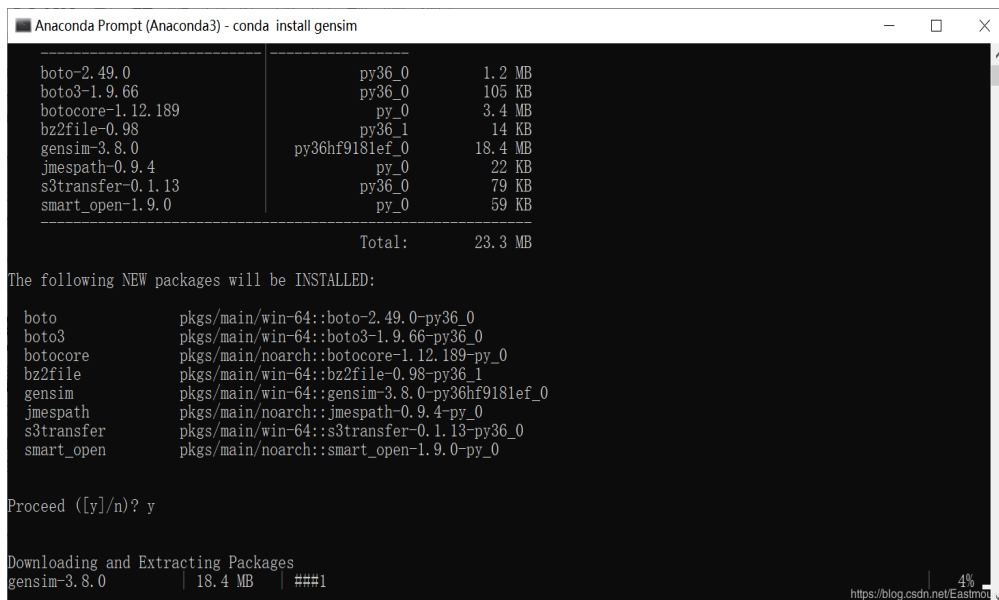
environment location: C:\Software\Program Software\Anaconda3\envs\tensorflow

added / updated specs:
- gensim

The following packages will be downloaded:

package                                     build
-----
boto-2.49.0                                py36_0                                1.2 MB

```



```

package                                     build
-----
boto-2.49.0                                py36_0                                1.2 MB
boto3-1.9.66                               py36_0                                105 KB
botocore-1.12.189                          py_0                                  3.4 MB
bz2file-0.98                               py36_1                                14 KB
gensim-3.8.0                               py36hf9181ef_0                       18.4 MB
jmespath-0.9.4                             py_0                                  22 KB
s3transfer-0.1.13                          py36_0                                79 KB
smart_open-1.9.0                           py_0                                  59 KB
Total:                                     23.3 MB

The following NEW packages will be INSTALLED:

boto                pkgs/main/win-64::boto-2.49.0-py36_0
boto3               pkgs/main/win-64::boto3-1.9.66-py36_0
botocore            pkgs/main/noarch::botocore-1.12.189-py_0
bz2file             pkgs/main/win-64::bz2file-0.98-py36_1
gensim              pkgs/main/win-64::gensim-3.8.0-py36hf9181ef_0
jmespath            pkgs/main/noarch::jmespath-0.9.4-py_0
s3transfer          pkgs/main/win-64::s3transfer-0.1.13-py36_0
smart_open          pkgs/main/noarch::smart_open-1.9.0-py_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
gensim-3.8.0        | 18.4 MB | ###1

```

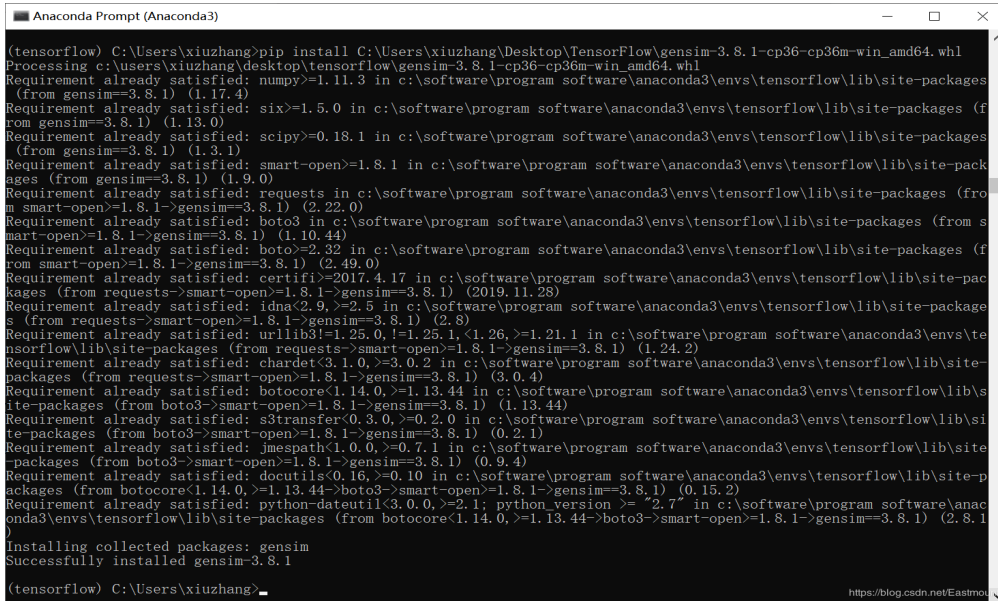
但是，由于anaconda的.org服务器在国外，下载速度很慢，提示错误“Anaconda An HTTP error occurred when trying to retrieve this URL.HTTP errors are often intermittent”。

- 解决方法一：从国内清华的镜像下载，参考文章：配置清华PIP镜像
- 解决方法二：从PYPI网站 (<https://pypi.org/project/gensim/#files>) 下载对应版本的 gensim，在再安装本地下载的.whl文件。

由于第一种方法一直失败，这里推荐读者尝试第二种方法，同时作者会将完整的实验环境上传供大家直接使用。

第三步，调用PIP安装本地gensim扩展包。


```
pip install C:\Users\xiuzhang\Desktop\TensorFlow\gensim-3.8.1-cp36-cp36m-
```



```

(tensorflow) C:\Users\xiuzhang>pip install C:\Users\xiuzhang\Desktop\TensorFlow\gensim-3.8.1-cp36-cp36m-win_amd64.whl
Processing c:\users\xiuzhang\desktop\tensorflow\gensim-3.8.1-cp36-cp36m-win_amd64.whl
Requirement already satisfied: numpy>=1.11.3 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-packages
  (from gensim==3.8.1) (1.17.4)
Requirement already satisfied: six>=1.5.0 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-packages (f
  rom gensim==3.8.1) (1.13.0)
Requirement already satisfied: scipy>=0.18.1 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-packages
  (from gensim==3.8.1) (1.3.1)
Requirement already satisfied: smart-open>=1.8.1 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-pack
  ages (from gensim==3.8.1) (1.9.0)
Requirement already satisfied: requests in c:\software\program software\anaconda3\envs\tensorflow\lib\site-packages (fro
  m smart-open=>1.8.1->gensim==3.8.1) (2.22.0)
Requirement already satisfied: boto3 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-packages (from s
  mart-open=>1.8.1->gensim==3.8.1) (1.10.44)
Requirement already satisfied: boto>=2.32 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-packages (f
  rom smart-open=>1.8.1->gensim==3.8.1) (2.49.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-pac
  kages (from requests->smart-open=>1.8.1->gensim==3.8.1) (2019.11.28)
Requirement already satisfied: idna<2.9,>=2.5 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-package
  s (from requests->smart-open=>1.8.1->gensim==3.8.1) (2.8)
Requirement already satisfied: urllib3<1.25.0,>=1.25.1,<1.26,>=1.21.1 in c:\software\program software\anaconda3\envs\te
  nsorflow\lib\site-packages (from requests->smart-open=>1.8.1->gensim==3.8.1) (1.24.2)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-
  packages (from requests->smart-open=>1.8.1->gensim==3.8.1) (3.0.4)
Requirement already satisfied: botocore<1.14.0,>=1.13.44 in c:\software\program software\anaconda3\envs\tensorflow\lib\s
  ite-packages (from boto3->smart-open=>1.8.1->gensim==3.8.1) (1.13.44)
Requirement already satisfied: s3transfer<0.3.0,>=0.2.0 in c:\software\program software\anaconda3\envs\tensorflow\lib\si
  te-packages (from boto3->smart-open=>1.8.1->gensim==3.8.1) (0.2.1)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in c:\software\program software\anaconda3\envs\tensorflow\lib\site
  -packages (from boto3->smart-open=>1.8.1->gensim==3.8.1) (0.9.4)
Requirement already satisfied: docutils<0.16,>=0.10 in c:\software\program software\anaconda3\envs\tensorflow\lib\site-p
  ackages (from botocore<1.14.0,>=1.13.44->boto3->smart-open=>1.8.1->gensim==3.8.1) (0.15.2)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in c:\software\program software\anac
  onda3\envs\tensorflow\lib\site-packages (from botocore<1.14.0,>=1.13.44->boto3->smart-open=>1.8.1->gensim==3.8.1) (2.8.1
  )
Installing collected packages: gensim
Successfully installed gensim-3.8.1

(tensorflow) C:\Users\xiuzhang>_

```

安装成功之后，开始编写我们的代码吧！

2.基础用法

第一步，调用random.choice随机生成数据集。

初始数据集为三个方向“人工智能”、“网络安全”和“网站开发”，然后调用choice函数随机选择某行数据，再从该行数据中4个词语随机挑选，最终形成1000行的数据集。

```

from random import choice
from gensim.models import Word2Vec

#-----数据集生成-----
# 定义初始数据集
ls_of_ls = [['Python', '大数据', '人工智能', 'Tensorflow'],
            ['网络安全', 'Web渗透', 'SQLMAP', 'Burpsuite'],
            ['网站开发', 'Java', 'MySQL', 'HTML5']]

# 真实项目中为数据集中文分词(jieba.cut)后的词列表
ls_of_words = []
for i in range(1000):
    ls = choice(ls_of_ls) #随机选择某行数据集
    ls_of_words.append([choice(ls) for _ in range(9, 15)])
print(ls_of_words)

```

输出部分结果如下所示，真实项目中会将中文预料分词，然后构建特征预料。

```
[['Web渗透', 'SQLMAP', 'Burpsuite', 'SQLMAP', 'Web渗透', '网络安全'],
 ['HTML5', 'MySQL', 'Java', 'MySQL', 'HTML5', 'MySQL'],
 ['Tensorflow', '大数据', '人工智能', 'Python', 'Tensorflow', '大数据'],
 ['Tensorflow', '人工智能', '大数据', 'Python', 'Python', '大数据'],
 ['网络安全', 'Burpsuite', '网络安全', 'SQLMAP', 'Burpsuite', '网络安全'],
 ['Java', 'Java', '网站开发', '网站开发', 'Java', 'HTML5'],
 ....
 ['人工智能', '人工智能', 'Python', 'Python', 'Tensorflow', 'Tensorflow'],
 ['HTML5', 'Java', '网站开发', '网站开发', 'MySQL', '网站开发'],
 ['Python', '大数据', 'Python', 'Tensorflow', 'Python', '大数据'],
 ['MySQL', 'MySQL', 'HTML5', '网站开发', 'MySQL', 'HTML5'],
 ['HTML5', 'MySQL', 'MySQL', 'Java', 'Java', '网站开发']
```

第二步，调用Word2Vec进行词向量训练，这里仅一个参数，即传入的数据集序列。

```
# 训练
model = Word2Vec(ls_of_words)
print(model)
```

其输出结果为模型，即Word2Vec(vocab=12, size=100, alpha=0.025)。参数含义如下：

- sentences：传入的数据集序列（list of lists of tokens），默认值为None
- size：词向量维数，默认值为100
- window：同句中当前词和预测词的最大距离，默认值为5
- min_count：最低词频过滤，默认值为5
- workers：线程数，默认值为3
- sg：模型参数，其值为0表示CBOW，值为1表示skip-gram，默认值为0
- hs：模型参数，其值为0表示负例采样，值为1表示层次softmax，默认值为0
- negative：负例样本数，默认值为5
- ns_exponent：用于形成负例样本的指数，默认值为0.75
- cbow_mean：上下文词向量参数，其值为0表示上下文词向量求和值，值为1表示上下文词向量平均值，默认值为1
- alpha：初始学习率，默认值为0.025
- min_alpha：最小学习率，默认值为0.0001

第三步，计算两个词语之间的相似度、某个词语与所有词语的相似度。

```
print(model.wv.similar_by_word('Tensorflow'))
print(model.wv.similarity('Web渗透', 'SQLMAP'))
```

输出结果如下图所示，可以发现“TensorFlow”最相关的词语包括“人工智能”、“Python”、“大数据”等，然后才是其他关键词，这就是所谓的基于上下文语义信息的相似度计算。在真实的数据集中，某个词如“人工智能”，经常和“CNN”同时出现，其相似度就较高；反之较低。通过该方法可以进行主题提取、文本分类、实体消歧等研究，比如将“体育”、“音乐”、“娱乐”、“学术”等不同主题新闻进行分类。

```
[('人工智能', 0.989766538143158),
 ('Python', 0.9859075546264648),
 ('大数据', 0.9857005476951599),
 ('网络安全', 0.9730607271194458),
 ('Java', 0.9701968431472778),
 ('HTML5', 0.9696391820907593),
 ('Web渗透', 0.9695969820022583),
 ('SQLMAP', 0.9685726165771484),
 ('Burpsuite', 0.9682992696762085),
 ('网站开发', 0.9662492871284485)]
0.9881845
```

第四步，显示词语并计算词向量矩阵及相似度。

```
#-----词矩阵计算-----
# 显示词
print("【显示词语】")
print(model.wv.index2word)
# 显示词向量矩阵
print("【词向量矩阵】")
vectors = model.wv.vectors
print(vectors)
print(vectors.shape)
# 显示四个词语最相关的相似度
print("【词向量相似度】")
for i in range(4):
    print(model.wv.similar_by_vector(vectors[i]))
```

输出结果如下所示，可以看到我们的12个词语，每个词语会表示成一个100维的词向量，最终形状为（12,100）。接着计算四个词语的相似度，第一个词语是“TensorFlow”，它与自身的相似度为1，与“人工智能”、“大数据”、“Python”等高度相似。

【显示词语】

['Tensorflow', 'MySQL', '人工智能', '大数据', 'Python', 'SQLMAP', 'HTML5',

【词向量矩阵】

```
[[-2.95917643e-03 -2.66699842e-03 -2.78887269e-03 ... -1.96327344e-02
  -8.24492238e-03  1.00784563e-02]
```

```
[-1.68114714e-03 -1.09694153e-02 -5.14865573e-03 ... -1.93562061e-02
  -8.51074420e-03  2.57991627e-02]
```

```
[-2.47456809e-03 -5.95887068e-05 -4.99962037e-03 ... -1.30442847e-02
  -1.27229355e-02  9.84096527e-03]
```

...

```
[ 6.16406498e-04 -2.90613738e-03  3.06741858e-04 ... -9.37678479e-03
  -5.20517444e-03  1.89058483e-02]
```

```
[-1.10806311e-02 -1.09268585e-02 -5.63549530e-03 ... -8.65904987e-03
  -4.51919530e-03  2.01134961e-02]
```

```
[-1.34018352e-02 -1.60384644e-03  2.59250798e-03 ... -1.38302138e-02
  -1.92157237e-03  1.76613722e-02]]
```

(12, 100)

【词向量相似度】

(('Tensorflow', 1.0), ('人工智能', 0.9916591644287109), ('大数据', 0.990448

(('MySQL', 1.0), ('网站开发', 0.9906301498413086), ('HTML5', 0.98995655775

(('人工智能', 0.9999999403953552), ('Tensorflow', 0.9916592836380005), ('大

(('大数据', 1.0), ('Tensorflow', 0.9904486536979675), ('人工智能', 0.990137

第五步，调用predict_output_word()函数预测新词，并计算器概率总和。

```
print("【预测新词】")
print(model.predict_output_word(['人工智能']))
total = sum(i[1] for i in model.predict_output_word(['人工智能']))
print('概率总和为%.2f' % total)
```

输出结果如下所示：

【预测新词】

```
[('网站开发', 0.08391691), ('网络安全', 0.08386225), ('SQLMAP', 0.08367824)
 ('HTML5', 0.08365326), ('Burpsuite', 0.083473735), ('Web渗透', 0.08326004)
 ('Tensorflow', 0.08323507), ('Java', 0.08317445), ('MySQL', 0.08317307),
 ('Python', 0.08307663)]
```

概率总和为0.83

完整代码为：

```

# -*- coding: utf-8 -*-
"""
Created on Mon Dec 23 17:48:50 2019
@author: xiuzhang Eastmount CSDN
"""

from random import choice
from gensim.models import Word2Vec

#-----数据集生成-----
# 定义初始数据集
ls_of_ls = [['Python', '大数据', '人工智能', 'Tensorflow'],
            ['网络安全', 'Web渗透', 'SQLMAP', 'Burpsuite'],
            ['网站开发', 'Java', 'MySQL', 'HTML5']]

# 真实项目中为数据集中文分词(jieba.cut)后的词列表
ls_of_words = []
for i in range(1000):
    ls = choice(ls_of_ls) #随机选择某行数据集
    ls_of_words.append([choice(ls) for _ in range(9, 15)])
print(ls_of_words)

#-----词向量训练-----
# 训练
model = Word2Vec(ls_of_words)
print(model)

#-----计算词语之间相似度-----
print(model.wv.similar_by_word('Tensorflow'))
print(model.wv.similarity('Web渗透', 'SQLMAP'))

#-----词矩阵计算-----
# 显示词
print("【显示词语】")
print(model.wv.index2word)
# 显示词向量矩阵
print("【词向量矩阵】")
vectors = model.wv.vectors
print(vectors)
print(vectors.shape)
# 显示四个词语最相关的相似度
print("【词向量相似度】")
for i in range(4):
    print(model.wv.similar_by_vector(vectors[i]))

#-----预测新词-----
print("【预测新词】")
print(model.predict_output_word(['人工智能']))

```

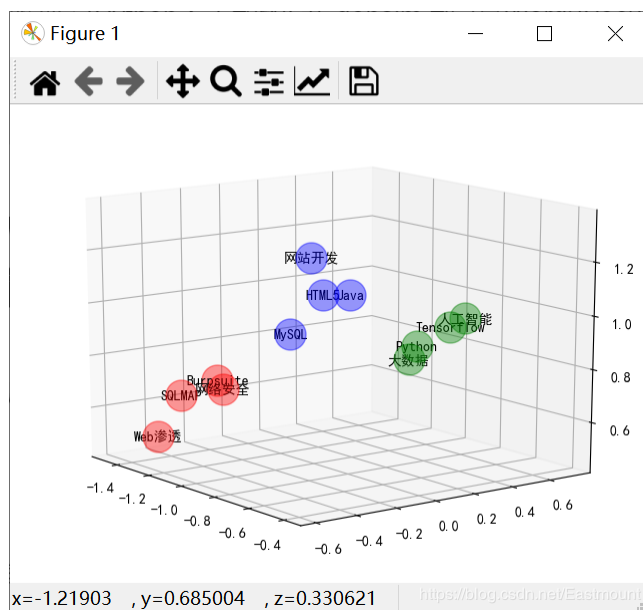


```
total = sum(i[1] for i in model.predict_output_word(['人工智能']))
print('概率总和为%.2f' % total)
```

写到这里，Word2Vec基础用法介绍完毕，接下来我们将结合具体案例分享中文短文本相似度的分析。同时，本文实验参考了很多yellow_python大神的文章，真心推荐大家学习，他的地址为：https://blog.csdn.net/Yellow_python。

三.Word2Vec+DBSCAN短文本聚类及可视化

首先我们用Word2Vec结合DBSCAN对上一部分的案例进行聚类分析及可视化展示，预测的类标为：[0 0 0 1 0 1 1 1 2 2 2 2]，对应的图如下所示，其显示的结果挺好的。



接着我们来看看完整的代码，详细过程见注释。

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
Created on Mon Dec 23 17:48:50 2019
```

```
@author: xiuzhang Eastmount CSDN
```

```
"""
```

```
from random import choice
from gensim.models import Word2Vec
from sklearn.cluster import DBSCAN
import matplotlib
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
```

```
#-----数据集生成-----
```

```
# 定义初始数据集
```

```
ls_of_ls = [['Python', '大数据', '人工智能', 'Tensorflow'],
```

```

['网络安全', 'Web渗透', 'SQLMAP', 'Burpsuite'],
['网站开发', 'Java', 'MySQL', 'HTML5']]

# 真实项目中为数据集中文分词(jieba.cut)后的词列表
ls_of_words = []
for i in range(2000):
    ls = choice(ls_of_ls) #随机选择某行数据集
    ls_of_words.append([choice(ls) for _ in range(9, 15)])
print(ls_of_words)

#-----词向量训练-----
# 训练 size词向量维数3 window预测距离7
model = Word2Vec(ls_of_words, size=3, window=7)
print(model)
# 提取词向量
vectors = [model[word] for word in model.wv.index2word]

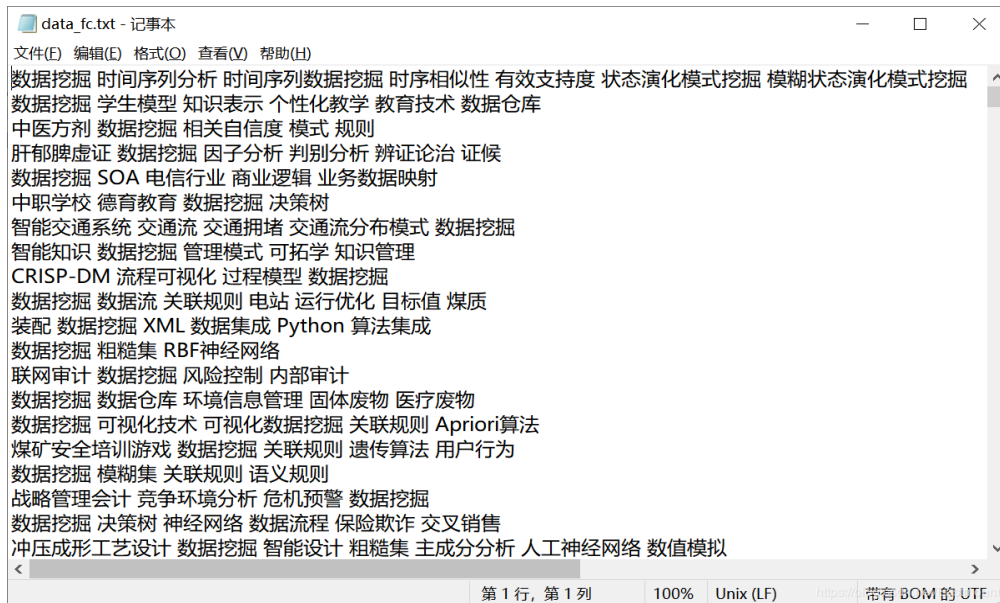
#-----词向量聚类-----
# 基于密度的DBSCAN聚类
labels = DBSCAN(eps=0.24, min_samples=3).fit(vectors).labels_
print(labels)

#-----可视化显示-----
plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文
matplotlib.rcParams['axes.unicode_minus'] = False # 显示负号
fig = plt.figure()
ax = mplot3d.Axes3D(fig) # 创建3d坐标轴
colors = ['red', 'blue', 'green', 'black']

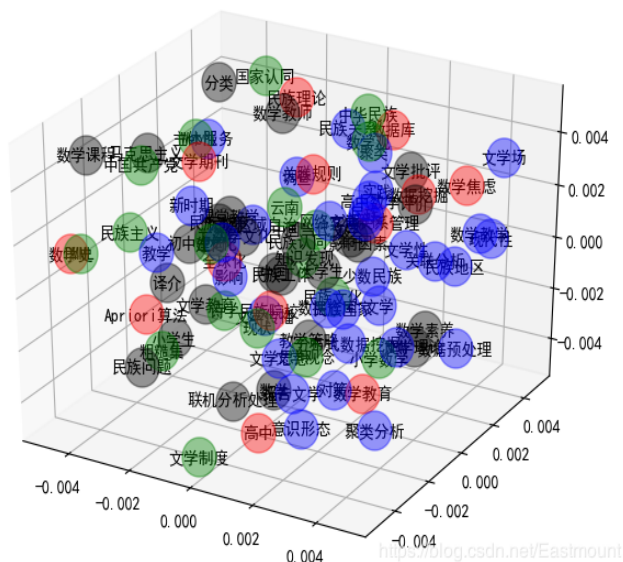
# 绘制散点图 词语 词向量 类标(颜色)
for word, vector, label in zip(model.wv.index2word, vectors, labels):
    ax.scatter(vector[0], vector[1], vector[2], c=colors[label], s=500, z=vector[2])
    ax.text(vector[0], vector[1], vector[2], word, ha='center', va='center')
plt.show()

```

假设我们从中国知网抓取了四个主题论文的关键词，包括数据挖掘、民族、数学、文学。每个主题包括220篇论文，其关键词均空格连接（已分词），如下图所示。



接着通过读取数据集，并转换成词向量序列进行聚类，最终输出结果如下图所示，效果不是很理想，后续深入分析。



完整代码如下图所示:

```
# -*- coding: utf-8 -*-
```

Created on Mon Dec 23 17:48:50 2019

@author: xiuzhang Eastmount CSDN

```
from gensim.models import Word2Vec
from sklearn.cluster import KMeans
import matplotlib
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
```

#-----加载语料-----

```
file name = "data_fc.txt"
```

```

ls_of_words = []
f = open(file_name, encoding='utf-8')
for lines in f.readlines():
    words = lines.strip().split(" ")
    #print(words)
    ls_of_words.append(words)
print(ls_of_words)

#-----词向量训练-----
# 训练 size词向量维数100 vocab单词200
model = Word2Vec(sentences=ls_of_words, size=100, window=10)
print(model)
# 提取词向量
vectors = [model[word] for word in model.wv.index2word]
print(len(model.wv.index2word))

#-----词向量聚类-----
# 基于KMeans 聚类
labels = KMeans(n_clusters=4).fit(vectors).labels_
print(labels)

#-----可视化显示-----
plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文
matplotlib.rcParams['axes.unicode_minus'] = False # 显示负号
fig = plt.figure()
ax = mplot3d.Axes3D(fig) # 创建3d坐标轴
colors = ['red', 'blue', 'green', 'black']
# 绘制散点图 词语 词向量 类标(颜色)
for word, vector, label in zip(model.wv.index2word, vectors, labels):
    ax.scatter(vector[0], vector[1], vector[2], c=colors[label], s=500, z=vector[2])
    ax.text(vector[0], vector[1], vector[2], word, ha='center', va='center')
plt.show()

```

四.Word2Vec计算《庆余年》中文短文本相似度

第一步，我们下载《庆余年》TXT小说，编码方式设置为“UTF-8”。



第二步，调用Jieba工具进行中文分词，安装调用“pip install jieba”。

```
C:\Software\Program Software\Python37\Scripts>pip install jieba
Collecting jieba
  Downloading https://files.pythonhosted.org/packages/71/46/c6f9179f73b818d5827202ad1c4a94e371a29473b7f043b736b4dab6b8cd/jieba-0.39.zip (7.3MB)
    |#####| 7.3MB 32kB/s
Installing collected packages: jieba
  Running setup.py install for jieba ... done
Successfully installed jieba-0.39
```

中文分词代码如下所示：

```
import os
import jieba
from gensim.models import Word2Vec
from gensim.models import word2vec

#----- 中文分词 -----
# 定义中文分词后文件名
file_name = "庆余年.txt"
cut_file = "庆余年_cut.txt"
# 文件读取操作
f = open(file_name, 'r', encoding='utf-8')
text = f.read()
# Jieba分词
new_text = jieba.cut(text, cut_all=False)      # 精确模式
# 过滤标点符号
str_out = ' '.join(new_text).replace(',', ' ').replace('。', ' ').replace('！', ' ').replace('“', ' ').replace('”', ' ').replace(':', ' ').replace('...', ' ').replace('-', ' ').replace('《', ' ').replace('》', ' ').replace('<', ' ').replace('>', ' ')
# 输出文件
fo = open(cut_file, 'w', encoding='utf-8')
# 写入操作
fo.write(str_out)
```



```
f.close()
fo.close()
```

第三步，调用Word2vec进行上下文语义相似度计算，代码如下。

这里的亮点是代码中判断是否训练，如果存在训练模型文件，则直接调用，下次

```
#-----训练模型-----
save_model_name = '庆余年.model'
# 判断训练的模型文件是否存在
if not os.path.exists(save_model_name):
    sentences = word2vec.Text8Corpus(cut_file)
    model = Word2Vec(sentences, size=200)
    model.save(save_model_name)
    # 二进制类型保存模型 后续直接使用
    model.wv.save_word2vec_format(save_model_name + ".bin", binary=True)
else:
    print('此训练模型已经存在，不用再次训练')

#-----预测结果-----
# 加载已训练好的模型
model = Word2Vec.load(save_model_name)

# 计算两个词的相似度/ 相关程度
res1 = model.wv.similarity("范闲", "林婉儿")
print(u"范闲和林婉儿的相似度为: ", res1, "\n")

# 计算某个词的相关词列表
res2 = model.wv.most_similar("范闲", topn=10) # 10个最相关的
print(u"和【范闲】最相关的词有: \n")
for item in res2:
    print(item[0], item[1])
print("-----\n")

# 计算某个词的相关词列表
res3 = model.wv.most_similar("五竹", topn=10) # 10个最相关的
print(u"和【五竹】最相关的词有: \n")
for item in res3:
    print(item[0], item[1])
print("-----\n")
```

最终输出结果如下所示：



庆余年.model

庆余年.
model.bin

庆余年.txt



庆余年_cut.txt

<https://blog.csdn.net/Eastmount>

此训练模型已经存在，不用再次训练
范闲和林婉儿的相似度为： 0.76498425

和【范闲】最相关的词有：

他 0.8427201509475708
婉儿 0.7699953317642212
林婉儿 0.7649842500686646
洪竹 0.7614920139312744
她 0.7607567310333252
夏栖飞 0.7595056891441345
言冰云 0.7514396905899048
杨万里 0.7514225840568542
明青达 0.7475594282150269
五竹 0.7399575710296631

和【五竹】最相关的词有：

肖恩 0.7810727953910828
言冰云 0.7754340171813965
他 0.7447681427001953
范闲 0.7399576902389526
她 0.7292275428771973
婉儿 0.7235128879547119
海棠 0.721315860748291
洪竹 0.7139796018600464
陈萍萍 0.7094859480857849
林婉儿 0.6981754302978516

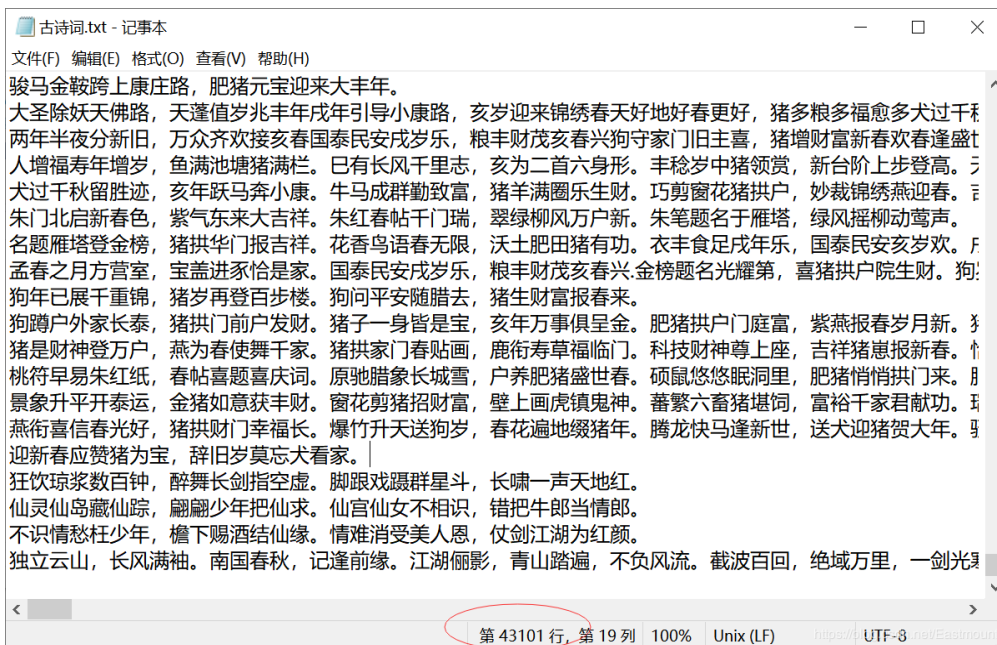
注：该部分思路参考了笑傲苍穹老师的思想，推荐大家学习：中文word2vec的python实现

五.Word2Vec写诗详解

最后补充一段Yellow_python大神的写诗博客，这里强烈推荐大家学习他的论文，AI和NLP系列每篇都是精华。博客地址：Python程序写诗【训练1分钟】古诗生成 - Yellow_python

基本流程如下：

- 首先，存在一个“古诗词.txt”数据集，包含了43101首诗，通过该数据集进行训练
- 接着，读取古诗词数据，通过 `ls_of_ls_of_c = [list(line.strip()) for line in f]` 代码将古诗词拆分成单个字，每首诗里面的字都有上下文
- 然后，调用Word2Vec进行训练，转换成词向量并保存模型
- 最后，输入标题进行补全，根据标题四个字进行语义查找，计算每个字的相似度，最终形成对应的诗句。核心代码为：`model.predict_output_word(poem[-self.window:], max(self.topn, len(poem) + 1))`



完整代码如下：

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Mon Dec 23 17:48:50 2019
@author: xiuzhang Eastmount CSDN
```

```
博客原地址: https://blog.csdn.net/Yellow_python/article/details/86726619
推荐大家学习Yellow_python大神的文章
"""
```

```
from gensim.models import Word2Vec # 词向量
from random import choice
from os.path import exists
```

```
class CONF:
```

```

path = '古诗词.txt'
window = 16          # 滑窗大小
min_count = 60       # 过滤低频字
size = 125           # 词向量维度
topn = 14            # 生成诗词的开放度
model_path = 'word2vec'

# 定义模型
class Model:
    def __init__(self, window, topn, model):
        self.window = window
        self.topn = topn
        self.model = model # 词向量模型
        self.chr_dict = model.wv.index2word # 字典

    """模型初始化"""
    @classmethod
    def initialize(cls, config):
        if exists(config.model_path):
            # 模型读取
            model = Word2Vec.load(config.model_path)
        else:
            # 语料读取
            with open(config.path, encoding='utf-8') as f:
                ls_of_ls_of_c = [list(line.strip()) for line in f]
            # 模型训练和保存
            model = Word2Vec(sentences=ls_of_ls_of_c, size=config.size, \
                             model.save(config.model_path))
            return cls(config.window, config.topn, model)

    """古诗词生成"""
    def poem_generator(self, title, form):
        filter = lambda lst: [t[0] for t in lst if t[0] not in [' ', ' ', '。']]
        # 标题补全
        if len(title) < 4:
            if not title:
                title += choice(self.chr_dict)
            for _ in range(4 - len(title)):
                similar_chr = self.model.similar_by_word(title[-1], self
                similar_chr = filter(similar_chr)
                char = choice([c for c in similar_chr if c not in title])
                title += char

        # 文本生成
        poem = list(title)
        for i in range(form[0]):
            for _ in range(form[1]):

```

```

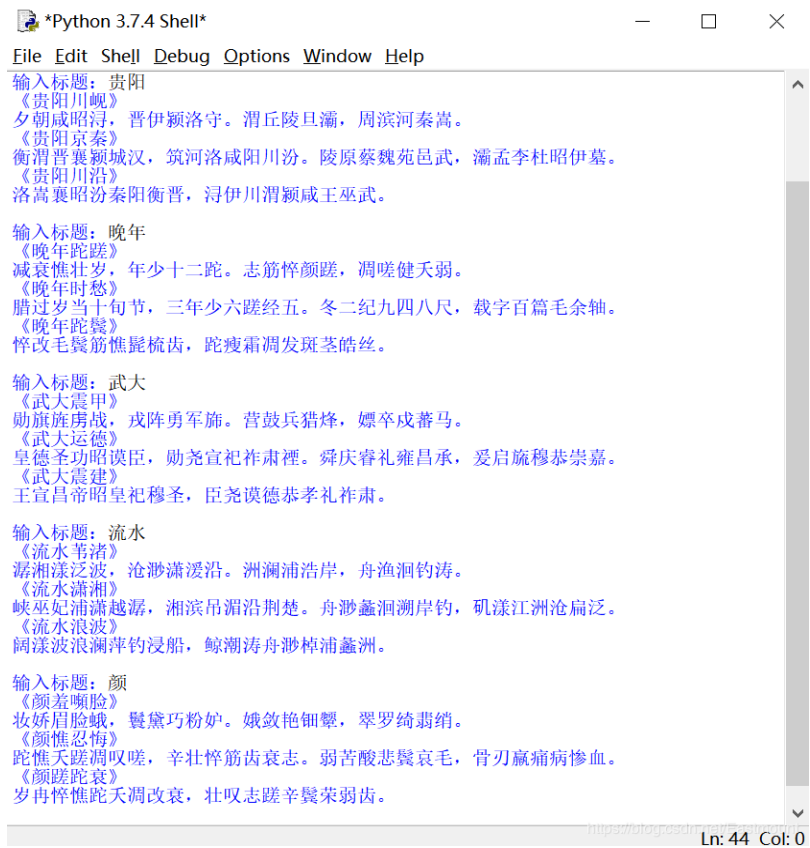
        predict_chr = self.model.predict_output_word(poem[-self.v
        predict_chr = filter(predict_chr)
        char = choice([c for c in predict_chr if c not in poem[-l
        poem.append(char)
        poem.append(', ' if i % 2 == 0 else '。')
    length = form[0] * (form[1] + 1)
    return '《%s》' % ''.join(poem[:length]) + '\n' + ''.join(poem[-

def main(config=CONF):
    form = {'五言绝句': (4, 5), '七言绝句': (4, 7), '对联': (2, 9)}
    m = Model.initialize(config)
    while True:
        title = input('输入标题: ').strip()
        try:
            poem = m.poem_generator(title, form['五言绝句'])
            print('%s' % poem) # red
            poem = m.poem_generator(title, form['七言绝句'])
            print('%s' % poem) # yellow
            poem = m.poem_generator(title, form['对联'])
            print('%s' % poem) # purple
            print()
        except:
            pass

if __name__ == '__main__':
    main()

```

输出结果如下图所示:



比如输入“武大”，输出的五言绝句、七严绝句如下所示，注意“勋旗旌虏战”这句话在我们的“古诗词.txt”中是找不到的，而是因为“勋”和“旗”存在相似度语义所致。

输入标题：武大
《武大震甲》
勋旗旌虏战，戎阵勇军旆。营鼓兵猎烽，嫖卒戍蕃马。
《武大运德》
皇德圣功昭谏臣，勋尧宣祀祚肃祿。舜庆睿礼雍昌承，爰启旒穆恭崇嘉。
《武大震建》
王宜昌帝昭皇祀穆圣，臣尧谟德恭孝礼祚肃。

六.总结

写到这里，这篇文章就讲解完毕，不知道您是否体会到了Word2Vec的强大功能。但如果您从事论文研究，您会发现它真是无处不在，从数据挖掘、自然语言处理大数据、人工智能，再到情感分析、图书情报、医学本体识别等，甚至恶意代码识别也有它的身影。但它写得诗也有辞藻堆砌的感觉，后续我们学到强化学习的时候，再对比它们的效果。

本文详细介绍了Word2Vec的基本原理，通过案例代码详细介绍了Word2Vec的用法。最后，希望这篇基础性文章对您有所帮助，如果文章中存在错误或不足之处，还请海涵~作为人工智能的菜鸟，我希望自己能不断进步并深入，后续将它应用于图像识别、网络安全、对抗样本等领域，指导大家撰写简单的学术论文，一起加油！

PS: 这是作者的第一个付费专栏，会非常用心的去撰写，希望能对得起读者的9块钱。本来只想设置1快的，但CSDN固定了价格。写了八年的免费文章，这也算知识付费的一个简单尝试吧！毕竟读博也不易，写文章也花费时间和精力，但作者更多的文章会免费分享。如果您购买了该专栏，有Python数据分析、图像处理、人工智能、网络安全的问题，我们都可以深入探讨，尤其是做研究的同学，共同进步~

(By:Eastmount 2019-12-23 晚上10点夜于珞珈山 <http://blog.csdn.net/eastmount/>)

闲谈:

最后希望大家帮我2019年CSDN博客之星投投票，每天可以投5票喔，谢谢大家！八年，在CSDN分享了410篇文章，15个专栏，400多万人次浏览，包括Python人工智能、数据挖掘、网络爬虫、图象处理、网络安全、JAVA网站、Android开发、LAMP/WAMP、C#网络编程、C++游戏、算法和数据结构、面试总结、人生感悟等。当然还有我和你的故事，感恩一路有你，感谢一路同行，希望通过编程分享帮助到更多人，也希望学成之后回贵州教更多学生。因为喜欢，所以分享，且看且珍惜，加油！等我四年学成归来~

投票地址: <http://m234140.nofollow.ax.mvote.cn/opage/ed8141a0-ed19-774b-6b0d-39c3aaf89dde.html?from=singlemessage>



作者theano人工智能系列:

[Python人工智能] 一.神经网络入门及theano基础代码讲解

[Python人工智能] 二.theano实现回归神经网络分析

[Python人工智能] 三.theano实现分类神经网络及机器学习基础

[Python人工智能] 四.神经网络和深度学习入门知识

[Python人工智能] 五.theano实现神经网络正规化Regularization处理

[Python人工智能] 六.神经网络的评价指标、特征标准化和特征选择

[Python人工智能] 七.加速神经网络、激励函数和过拟合

参考文献:

[1] word2vec词向量训练及中文文本相似度计算 - 作者的文章

[2] 《Word2vec的核心架构及其应用》·熊富林, 邓怡豪, 唐晓晟·北邮2015年

[3] 《Word2vec的工作原理及应用探究》·周练·西安电子科技大学2014年

[4] word2vec——高效word特征求取 - 推荐Rachel-Zhang大神文章

[5] Deep Learning in NLP (一) 词向量和语言模型 - licstar大神

[6] gensim词向量Word2Vec - Yellow_python (强推)

[7] word2vec函数参数 - 冥更

[8] NLP之词向量: 利用word2vec对20类新闻文本数据集进行词向量训练、测试(某个单词的相关词汇) - 一个处女座的程序猿

[9] gensim中word2vec python源码理解 (一) 初始化构建单词表 - ForcedOverflow

[10] <https://github.com/AryeYellow/PyProjects>

[11] Python程序写诗【训练1分钟】古诗生成 - Yellow_python

[12] 机器学习100问|Word2Vec是如何工作的? 它和LDA有什么区别与联系?

[13] <https://github.com/eastmountyxz/AI-for-TensorFlow>

[14] windows下使用word2vec训练维基百科中文语料全攻略! (三) - 文哥的学习日记

[15] NLP之——Word2Vec详解 - 郭耀华

[16] 中文word2vec的python实现 - 笑傲苍穹0