# 如何准备电影评论数据以进行情感分析 如何准备电影评论数据以进行情感分析 原文链接： 每个问题的文本数据准备都不同。 准备工作从简单的步骤开始，例如加载数据，但是对于与您正在使用的数据非常相关的清理任务，很快就会变得困难。 从原始数据到准备好建模的步骤，您需要有关从何处开始以及以什么顺序工作的帮助。 在本教程中，您将逐步了解如何准备电影评论文本数据以进行情感分析。 完成本教程后，您将知道： 如何加载和清除文本数据以删除标点符号和其他非单词。 如何开发词汇表，定制词汇表并将其保存到文件中。

原文链接：[How to Prepare Movie Review Data for Sentiment Analysis](https://machinelearningmastery.com/prepare-movie-review-data-sentiment-analysis/?from=hackcv&hmsr=hackcv.com&utm_medium=hackcv.com&utm_source=hackcv.com)

每个问题的文本数据准备都不同。

准备工作从简单的步骤开始，例如加载数据，但是对于与您正在使用的数据非常相关的清理任务，很快就会变得困难。 从原始数据到准备好建模的步骤，您需要有关从何处开始以及以什么顺序工作的帮助。

在本教程中，您将逐步了解如何准备电影评论文本数据以进行情感分析。

完成本教程后，您将知道：

* 如何加载和清除文本数据以删除标点符号和其他非单词。
* 如何开发词汇表，定制词汇表并将其保存到文件中。
* 如何使用清洁和预定义的词汇准备电影评论，并将其保存到可供建模的新文件中。  
  如何使用清洁和预定义的词汇准备电影评论，并将其保存到可供建模的新文件中。  
  让我们开始吧。  
  2017年10月更新：修复了跳过不匹配文件时的一个小错误，感谢Jan Zett。  
  2017年12月更新：修复了完整示例中的小错字，感谢Ray和Zain。  
  如何准备电影评论数据以进行情感分析 摄，保留一些权利。  
  教程概述  
  本教程分为5个部分。 他们是：  
  电影评论数据集  
  加载文本数据  
  清除文本数据  
  训练词汇量  
  保存准备的数据  
  在文本数据深度学习方面需要帮助吗？  
  立即参加我的7天免费电子邮件崩溃课程（包含代码）。

让我们开始吧。

* **2017年10月更新**：修复了跳过不匹配文件时的一个小错误，感谢Jan Zett。
* **2017年12月更新**：修复了完整示例中的小错字，感谢Ray和Zain。

如何准备电影评论数据以进行情感分析 [肯尼思·鲁](https://www.flickr.com/photos/toasty/1125019024/)摄，保留一些权利。

## 教程概述

本教程分为5个部分。 他们是：

1. 电影评论数据集
2. 加载文本数据
3. 清除文本数据
4. 训练词汇量
5. 保存准备的数据

### 在文本数据深度学习方面需要帮助吗？

立即参加我的7天免费电子邮件崩溃课程（包含代码）。

单击以注册，并获得该课程的免费PDF电子书版本。  
单击以注册，并获得该课程的免费PDF电子书版本。  
1. 电影评论数据集  
电影评论数据是Bo Pang和Lillian Lee在2000年代初期从imdb.com网站检索的电影评论的集合。 收集了这些评论，并将其作为他们对自然语言处理的研究的一部分提供。  
这些评论最初于2002年发布，但于2004年发布了更新和清理的版本，称为“ \* v2.0 \*”。  
该数据集由从托管的rec.arts.movies.reviews新闻组的存档中抽取的1,000条正面和负面电影评论组成。 作者将此数据集称为“ 极性数据集”。  
我们的数据包含2002年之前撰写的1000条正面评论和1000条负面评论，每个类别每位作者的评论上限为20（总共312作者）。 我们将此语料库称为极性数据集。

[Start Your FREE Crash-Course Now](https://machinelearningmastery.lpages.co/leadbox/144855173f72a2%3A164f8be4f346dc/5655638436741120/)

## 1. 电影评论数据集

电影评论数据是Bo Pang和Lillian Lee在2000年代初期从imdb.com网站检索的电影评论的集合。 收集了这些评论，并将其作为他们对自然语言处理的研究的一部分提供。

这些评论最初于2002年发布，但于2004年发布了更新和清理的版本，称为“ \* v2.0 \*”。

该数据集由从[IMDB](http://reviews.imdb.com/Reviews)托管的rec.arts.movies.reviews新闻组的存档中抽取的1,000条正面和负面电影评论组成。 作者将此数据集称为“ *极性数据集*”。

我们的数据包含2002年之前撰写的1000条正面评论和1000条负面评论，每个类别每位作者的评论上限为20（总共312作者）。 我们将此语料库称为极性数据集。

— , 2004.  
— , 2004.  
数据已经进行了一些清理，例如：  
数据集仅包含英文评论。  
所有文字均已转换为小写。  
标点符号周围有空格，例如句号，逗号和方括号。  
文本已被分成每行一个句子。  
该数据已用于一些相关的自然语言处理任务。 对于分类，经典模型（例如支持向量机）在数据上的性能在高70％至低80％（例如78％至82％）的范围内。  
进行10倍交叉验证后，更复杂的数据准备可能会看到高达86％的结果。 如果我们希望在现代方法的实验中使用此数据集，那么这将使我们处于80年代中期到中期的水平。

数据已经进行了一些清理，例如：

* 数据集仅包含英文评论。
* 所有文字均已转换为小写。
* 标点符号周围有空格，例如句号，逗号和方括号。
* 文本已被分成每行一个句子。

该数据已用于一些相关的自然语言处理任务。 对于分类，经典模型（例如支持向量机）在数据上的性能在高70％至低80％（例如78％至82％）的范围内。

进行10倍交叉验证后，更复杂的数据准备可能会看到高达86％的结果。 如果我们希望在现代方法的实验中使用此数据集，那么这将使我们处于80年代中期到中期的水平。

…根据下游极性分类器的选择，我们可以实现统计学上显着的改进（从82.8％到86.4％）  
…根据下游极性分类器的选择，我们可以实现统计学上显着的改进（从82.8％到86.4％）  
— , 2004.  
您可以从此处下载数据集：  
(review\_polarity.tar.gz, 3MB)  
解压缩文件后，您将拥有一个名为“ txt\_sentoken”的目录，该目录包含两个子目录，分别包含否定和肯定评论的文本“ neg”和“ pos”。 评论每个文件存储一个，命名规则为cv000至cv999，分别用于neg和pos。  
接下来，让我们看一下加载文本数据。  
2. 载入文本数据  
在本节中，我们将研究加载单个文本文件，然后处理文件目录。

— [A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts](http://xxx.lanl.gov/abs/cs/0409058), 2004.

您可以从此处下载数据集：

* [Movie Review Polarity Dataset](http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz) (review\_polarity.tar.gz, 3MB)

解压缩文件后，您将拥有一个名为“ *txt\_sentoken*”的目录，该目录包含两个子目录，分别包含否定和肯定评论的文本“ *neg*”和“ *pos*”。 评论每个文件存储一个，命名规则为*cv000*至*cv999*，分别用于neg和pos。

接下来，让我们看一下加载文本数据。

## 2. 载入文本数据

在本节中，我们将研究加载单个文本文件，然后处理文件目录。

我们将假定已下载审阅数据并在当前工作目录的“ txt\_sentoken”文件夹中提供该审阅数据。  
我们将假定已下载审阅数据并在当前工作目录的“ txt\_sentoken”文件夹中提供该审阅数据。  
我们可以通过打开单个文本文件，读取ASCII文本并关闭文件来加载它。 这是标准的文件处理内容。 例如，我们可以按如下方式加载第一个负面评论文件“ cv000\_29416.txt”：  
# load one filefilename = 'txt\_sentoken/neg/cv000\_29416.txt'# open the file as read onlyfile = open(filename, 'r')# read all texttext = file.read()# close the filefile.close()  
这样会将文档加载为ASCII并保留所有空白，例如换行。  
我们可以将其转换为一个名为load\_doc()的函数，该函数将文档的文件名加载并返回文本。

我们可以通过打开单个文本文件，读取ASCII文本并关闭文件来加载它。 这是标准的文件处理内容。 例如，我们可以按如下方式加载第一个负面评论文件“ *cv000\_29416.txt*”：

# load one file  
filename = 'txt\_sentoken/neg/cv000\_29416.txt'  
# open the file as read only  
file = open(filename, 'r')  
# read all text  
text = file.read()  
# close the file  
file.close()

这样会将文档加载为ASCII并保留所有空白，例如换行。

我们可以将其转换为一个名为load\_doc()的函数，该函数将文档的文件名加载并返回文本。

# load doc into memory  
def load\_doc(filename):   
 # open the file as read only  
 file = open(filename, 'r')  
 # read all text   
 text = file.read()  
 # close the file  
 file.close()  
 return text  
# load doc into memorydef load\_doc(filename): # open the file as read only file = open(filename, 'r') # read all text text = file.read() # close the file file.close() return text  
我们有两个目录，每个目录包含1000个文档。 我们可以依次使用以下命令依次处理每个目录： , 然后依次加载每个文件。  
例如，我们可以使用\* load\_doc()\*函数在负目录中加载每个文档，以进行实际加载。  
from os import listdir # load doc into memorydef load\_doc(filename): # open the file as read only file = open(filename, 'r') # read all text text = file.read() # close the file file.close() return text # specify directory to loaddirectory = 'txt\_sentoken/neg'# walk through all files in the folderfor filename in listdir(directory): # skip files that do not have the right extension if not filename.endswith(".txt"): continue # create the full path of the file to open path = directory + '/' + filename # load document doc = load\_doc(path) print('Loaded %s' % filename)

我们有两个目录，每个目录包含1000个文档。 我们可以依次使用以下命令依次处理每个目录： [listdir() function](https://docs.python.org/3/library/os.html#os.listdir), 然后依次加载每个文件。

例如，我们可以使用\* load\_doc()\*函数在负目录中加载每个文档，以进行实际加载。

from os import listdir   
# load doc into memory  
def load\_doc(filename):   
 # open the file as read only   
 file = open(filename, 'r')   
 # read all text text = file.read()   
 # close the file   
 file.close()   
 return text   
# specify directory to load  
directory = 'txt\_sentoken/neg'  
# walk through all files in the folder  
for filename in listdir(directory):   
 # skip files that do not have the right extension   
 if not filename.endswith(".txt"):   
 continue   
 # create the full path of the file to open   
 path = directory + '/' + filename   
 # load document   
 doc = load\_doc(path)   
 print('Loaded %s' % filename)

运行此示例将在加载每个评论后打印其文件名。  
运行此示例将在加载每个评论后打印其文件名。  
...Loaded cv995\_23113.txtLoaded cv996\_12447.txtLoaded cv997\_5152.txtLoaded cv998\_15691.txtLoaded cv999\_14636.txt  
我们也可以将文档的处理转换为一个函数，稍后将其用作模板以开发用于清理文件夹中所有文档的函数。 例如，下面我们定义一个\* process\_docs()\*函数来执行相同的操作。  
from os import listdir # load doc into memorydef load\_doc(filename): # open the file as read only file = open(filename, 'r') # read all text text = file.read() # close the file file.close() return text # load all docs in a directorydef process\_docs(directory): # walk through all files in the folder for filename in listdir(directory): # skip files that do not have the right extension if not filename.endswith(".txt"): continue # create the full path of the file to open path = directory + '/' + filename # load document doc = load\_doc(path) print('Loaded %s' % filename) # specify directory to load directory = 'txt\_sentoken/neg'process\_docs(directory)

...Loaded cv995\_23113.txt  
Loaded cv996\_12447.txt  
Loaded cv997\_5152.txt  
Loaded cv998\_15691.txt  
Loaded cv999\_14636.txt

我们也可以将文档的处理转换为一个函数，稍后将其用作模板以开发用于清理文件夹中所有文档的函数。 例如，下面我们定义一个\* process\_docs()\*函数来执行相同的操作。

from os import listdir   
# load doc into memory  
def load\_doc(filename):   
 # open the file as read only   
 file = open(filename, 'r')   
 # read all text   
 text = file.read()   
 # close the file   
 file.close()   
 return text   
# load all docs in a directory  
def process\_docs(directory):   
# walk through all files in the folder   
for filename in listdir(directory):   
# skip files that do not have the right extension   
if not filename.endswith(".txt"):   
 continue   
 # create the full path of the file to open   
 path = directory + '/' + filename   
 # load document   
 doc = load\_doc(path)   
 print('Loaded %s' % filename)   
 # specify directory to load  
 directory = 'txt\_sentoken/neg'process\_docs(directory)

现在，我们知道了如何加载电影评论文本数据，让我们来看看如何清理它。  
现在，我们知道了如何加载电影评论文本数据，让我们来看看如何清理它。  
3. 清洗文本数据  
In this section, we will look at what data cleaning we might want to do to the movie review data.  
We will assume that we will be using a bag-of-words model or perhaps a word embedding that does not require too much preparation.  
Split into Tokens  
First, let’s load one document and look at the raw tokens split by white space. We will use the load\_doc() function developed in the previous section. We can use the split() function to split the loaded document into tokens separated by white space.

## 3. 清洗文本数据

In this section, we will look at what data cleaning we might want to do to the movie review data.

We will assume that we will be using a bag-of-words model or perhaps a word embedding that does not require too much preparation.

Split into Tokens

First, let’s load one document and look at the raw tokens split by white space. We will use the *load\_doc()* function developed in the previous section. We can use the *split()* function to split the loaded document into tokens separated by white space.

|  |  |
| --- | --- |
| 12345678910111213141516 1234567891011213141516 | # load doc into memorydef load\_doc(filename): # open the file as read only file = open(filename, ‘r’) # read all text text = file.read() # close the file file.close() return text # load the documentfilename = ’txt\_sentoken/neg/cv000\_29416.txt’text = load\_doc(filename)# split into tokens by white spacetokens = text.split()print(tokens) #将文档加载到memorydef加载到文档（文件名）：#以只读文件的形式打开文件=打开（文件名，'r'）#读取所有文本文本=文件。读取（）#关闭文件。关闭（）返回文本#加载文档文件名='txt\_sentoken/neg/cv000\_29416.txt'text=加载到文档（文件名）#通过白色空格标记=文本。拆分（）打印（标记） |
|  |  |

Running the example gives a nice long list of raw tokens from the document.  
运行该示例会给出一个很长的文档原始标记列表。

|  |  |
| --- | --- |
| 12 德意志北方银行 | …‘years’, ‘ago’, ‘and’, ‘has’, ‘been’, ‘sitting’, ‘on’, ‘the’, ‘shelves’, ‘ever’, ‘since’, ‘.’, ‘whatever’, ‘.’, ‘.’, ‘.’, ‘skip’, ‘it’, ‘!’, “where’s”, ‘joblo’, ‘coming’, ‘from’, ‘?’, ‘a’, ‘nightmare’, ‘of’, ‘elm’, ‘street’, ‘3’, ‘(’, ‘7/10’, ‘)’, ‘-’, ‘blair’, ‘witch’, ‘2’, ‘(’, ‘7/10’, ‘)’, ‘-’, ‘the’, ‘crow’, ‘(’, ‘9/10’, ‘)’, ‘-’, ‘the’, ‘crow’, ‘:’, ‘salvation’, ‘(’, ‘4/10’, ‘)’, ‘-’, ‘lost’, ‘highway’, ‘(’, ‘10/10’, ‘)’, ‘-’, ‘memento’, ‘(’, ‘10/10’, ‘)’, ‘-’, ‘the’, ‘others’, ‘(’, ‘9/10’, ‘)’, ‘-’, ‘stir’, ‘of’, ‘echoes’, ‘(’, ‘8/10’, ‘)’] “…”年“，”以前“，”已经“，”一直“，”坐着“，”在“，”架子“，”从“，”以后“，”无论“，”什么“，”什么“，”什么“，”什么“，”什么“，”跳过“，”它“，”！“，”哪里“，”joblo“，”coming“，”from“，”呢？“，”a“，”噩梦“，”榆树“，”街道“，”3“，”3“，”7/10“，”）“，”-“，”布莱尔“，”女巫“，”2“，”7/10“，”）“，”-“，”乌鸦“，”乌鸦“，”9/10“，”）“，”-“，”乌鸦“，”乌鸦“：”，”拯救“，”4/10“，”，“-”，“丢失”，“高速路”，“纪念品”，“纪念品”，“纪念品”，“纪念品”，“纪念品”，“纪念品”，“纪念品”，“纪念品”，“纪念品 |
|  |  |

Just looking at the raw tokens can give us a lot of ideas of things to try, such as:  
只要看看原始的代币就可以给我们很多尝试的想法，比如：

* Remove punctuation from words (e.g. ‘what’s’).  
  删除单词中的标点符号（例如“what's”）。
* Removing tokens that are just punctuation (e.g. ‘-‘).  
  删除只是标点符号的标记（例如“-”）。
* Removing tokens that contain numbers (e.g. ’10/10′).  
  移除包含数字的标记（例如“10/10”）。
* Remove tokens that have one character (e.g. ‘a’).  
  移除具有一个字符（例如“a”）的标记。
* Remove tokens that don’t have much meaning (e.g. ‘and’)  
  移除没有太多意义的标记（例如“and”）

Some ideas:  
一些想法：

* We can filter out punctuation from tokens using the string translate() function.  
  我们可以使用string translate（）函数从标记中筛选出标点符号。
* We can remove tokens that are just punctuation or contain numbers by using an isalpha()check on each token.  
  我们可以通过对每个标记使用isalpha（）检查来删除只是标点符号或包含数字的标记。
* We can remove English stop words using the list loaded using NLTK.  
  我们可以使用NLTK加载的列表删除英语停止词。
* We can filter out short tokens by checking their length.  
  我们可以通过检查短令牌的长度来过滤它们。

Below is an updated version of cleaning this review.  
下面是清理此评论的更新版本。

|  |  |
| --- | --- |
| 1234567891011121314151617181920212223242526272829 123456789101121314151617181920212223242526272829 | from nltk.corpus import stopwordsimport string # load doc into memorydef load\_doc(filename): # open the file as read only file = open(filename, ‘r’) # read all text text = file.read() # close the file file.close() return text # load the documentfilename = ‘txt\_sentoken/neg/cv000\_29416.txt’text = load\_doc(filename)# split into tokens by white spacetokens = text.split()# remove punctuation from each tokentable = str.maketrans(’‘,’‘, string.punctuation)tokens = [w.translate(table) for w in tokens]# remove remaining tokens that are not alphabetictokens = [word for word in tokens if word.isalpha()]# filter out stop wordsstop\_words = set(stopwords.words(’english’))tokens = [w for w in tokens if not w in stop\_words]# filter out short tokenstokens = [word for word in tokens if len(word) > 1]print(tokens) 从nltk.corpus导入stopwordsimport字符串#将文档加载到memorydef load \_doc（filename）：#将文件作为只读文件打开=打开（filename，'r'）#读取所有文本文本=file.read（）#关闭文件file.close（）返回文本#加载documentfilename='txt\_sentoken/neg/cv000\_29416.txt'text=load\_doc（filename）#按空白标记拆分为标记=text.split（）#从每个标记表中删除标点符号=str.maketrans（''，string.spoontion）tokens=[w.translate（table）for w in tokens]#删除不按字母顺序的剩余tokens=[word for word in tokens if word.isalpha（）]ţ过滤掉stop words stop words=set（stopwords.words（'english'）tokens=[w for w in tokens if not w in stopţwords]ţ过滤掉short tokens stokens=[words in tokens iflen（word）>1]打印（标记） |
|  |  |

Running the example gives a much cleaner looking list of tokens  
运行该示例会给出一个外观更清晰的令牌列表

|  |  |
| --- | --- |
| 12 德意志北方银行 | …‘explanation’, ‘craziness’, ‘came’, ‘oh’, ‘way’, ‘horror’, ‘teen’, ‘slasher’, ‘flick’, ‘packaged’, ‘look’, ‘way’, ‘someone’, ‘apparently’, ‘assuming’, ‘genre’, ‘still’, ‘hot’, ‘kids’, ‘also’, ‘wrapped’, ‘production’, ‘two’, ‘years’, ‘ago’, ‘sitting’, ‘shelves’, ‘ever’, ‘since’, ‘whatever’, ‘skip’, ‘wheres’, ‘joblo’, ‘coming’, ‘nightmare’, ‘elm’, ‘street’, ‘blair’, ‘witch’, ‘crow’, ‘crow’, ‘salvation’, ‘lost’, ‘highway’, ‘memento’, ‘others’, ‘stir’, ‘echoes’] “…”解释“，”疯狂“，”来了“，”哦“，”方式“，”恐怖“，”青少年“，”斜杠“，”电影“，”包装“，”看“，”方式“，”某人“，”显然“，”假设“，”类型“，”仍然“，”热门“，”孩子“，”也“，”包装“，”生产“，”两年“，”以前“，”，“坐着”、“架子”、“曾经”、“自从”、“随便”、“跳过”、“哪里”、“joblo”、“来了”、“噩梦”、“榆树”、“街道”、“布莱尔”、“女巫”、“乌鸦”、“救赎”、“迷路”、“纪念品”、“其他”、“搅拌”、“回声”] |
|  |  |

We can put this into a function called clean\_doc() and test it on another review, this time a positive review.  
我们可以将其放入一个名为clean\_doc（）的函数中，并在另一个评审中测试它，这次是一个肯定的评审。

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031323334 12345678910112131415161718192021222324252627293031323334 | from nltk.corpus import stopwordsimport string # load doc into memorydef load\_doc(filename): # open the file as read only file = open(filename, ‘r’) # read all text text = file.read() # close the file file.close() return text # turn a doc into clean tokensdef clean\_doc(doc): # split into tokens by white space tokens = doc.split() # remove punctuation from each token table = str.maketrans(’‘,’‘, string.punctuation) tokens = [w.translate(table) for w in tokens] # remove remaining tokens that are not alphabetic tokens = [word for word in tokens if word.isalpha()] # filter out stop words stop\_words = set(stopwords.words(’english’)) tokens = [w for w in tokens if not w in stop\_words] # filter out short tokens tokens = [word for word in tokens if len(word) > 1] return tokens # load the documentfilename = ’txt\_sentoken/pos/cv000\_29590.txt’text = load\_doc(filename)tokens = clean\_doc(text)print(tokens) from nltk.corpus import stopwordsimport string # load doc into memorydef load\_doc(filename): # open the file as read only file = open(filename, ‘r’) # read all text text = file.read() # close the file file.close() return text # turn a doc into clean tokensdef clean\_doc(doc): # split into tokens by white space tokens = doc.split() # remove punctuation from each token table = str.maketrans(’‘,’‘, string.punctuation) tokens = [w.translate(table) for w in tokens] # remove remaining tokens that are not alphabetic tokens = [word for word in tokens if word.isalpha()] # filter out stop words stop\_words = set(stopwords.words(’english’)) tokens = [w for w in tokens if not w in stop\_words] # filter out short tokens tokens = [word for word in tokens if len(word) > 1] return tokens # load the documentfilename = ’txt\_sentoken/pos/cv000\_29590.txt’text = load\_doc(filename)tokens = clean\_doc(text)print(tokens) |
|  |  |

Again, the cleaning procedure seems to produce a good set of tokens, at least as a first cut.  
同样，清理过程似乎产生了一组好的标记，至少作为第一次切割。

|  |  |
| --- | --- |
| 12 12 | …‘comic’, ‘oscar’, ‘winner’, ‘martin’, ‘childs’, ‘shakespeare’, ‘love’, ‘production’, ‘design’, ‘turns’, ‘original’, ‘prague’, ‘surroundings’, ‘one’, ‘creepy’, ‘place’, ‘even’, ‘acting’, ‘hell’, ‘solid’, ‘dreamy’, ‘depp’, ‘turning’, ‘typically’, ‘strong’, ‘performance’, ‘deftly’, ‘handling’, ‘british’, ‘accent’, ‘ians’, ‘holm’, ‘joe’, ‘goulds’, ‘secret’, ‘richardson’, ‘dalmatians’, ‘log’, ‘great’, ‘supporting’, ‘roles’, ‘big’, ‘surprise’, ‘graham’, ‘cringed’, ‘first’, ‘time’, ‘opened’, ‘mouth’, ‘imagining’, ‘attempt’, ‘irish’, ‘accent’, ‘actually’, ‘wasnt’, ‘half’, ‘bad’, ‘film’, ‘however’, ‘good’, ‘strong’, ‘violencegore’, ‘sexuality’, ‘language’, ‘drug’, ‘content’] “…”漫画“，”奥斯卡“，”获奖者“，”马丁“，”儿童“，”莎士比亚“，”爱情“，”制作“，”设计“，”转身“，”原创“，”布拉格“，”环境“，”一个“，”恐怖“，”地方“，”偶数“，”表演“，”地狱“，”固体“，”梦幻“，”德普“，”转身“，”典型“，“强”、“表现”、“灵巧”、“处理”、“英式”、“重音”、“ian”、“holm”、“joe”、“goulds”、“secret”、“richardson”、“dalmatians”、“log”、“great”、“supporting”、“roles”、“big”、“surprise”、“graham”、“cringed”、“first”、“time”、“opened”，‘嘴巴’、‘想象’、‘尝试’、‘爱尔兰语’、‘口音’、‘实际’、‘wasnt’、‘一半’、‘坏’、‘电影’、‘然而’、‘好’、‘强’、‘暴力’、‘性’、‘语言’、‘毒品’、‘内容’] |
|  |  |

There are many more cleaning steps we could take and I leave them to your imagination.  
我们还可以采取更多的清洁措施，让你想象。

Next, let’s look at how we can manage a preferred vocabulary of tokens.  
接下来，让我们看看如何管理标记的首选词汇表。

## 4. Develop Vocabulary 四。发展词汇

When working with predictive models of text, like a bag-of-words model, there is a pressure to reduce the size of the vocabulary.  
当使用文本的预测模型时，比如一个单词袋模型，会有一个减少词汇量的压力。

The larger the vocabulary, the more sparse the representation of each word or document.  
词汇量越大，每个单词或文档的表示就越稀疏。

A part of preparing text for sentiment analysis involves defining and tailoring the vocabulary of words supported by the model.  
为情感分析准备文本的一部分涉及定义和裁剪模型支持的词汇。

We can do this by loading all of the documents in the dataset and building a set of words. We may decide to support all of these words, or perhaps discard some. The final chosen vocabulary can then be saved to file for later use, such as filtering words in new documents in the future.  
我们可以通过加载数据集中的所有文档并构建一组单词来实现这一点。我们可以决定支持所有这些话，或者放弃一些。最后选定的词汇表可以保存到文件中供以后使用，例如将来在新文档中筛选单词。

We can keep track of the vocabulary in a , which is a dictionary of words and their count with some additional convenience functions.  
我们可以跟踪a中的词汇，这是一个词汇词典，它提供了一些额外的方便功能，并对词汇进行计数。

We need to develop a new function to process a document and add it to the vocabulary. The function needs to load a document by calling the previously developed load\_doc() function. It needs to clean the loaded document using the previously developed clean\_doc() function, then it needs to add all the tokens to the Counter, and update counts. We can do this last step by calling the update() function on the counter object.  
我们需要开发一个新函数来处理文档并将其添加到词汇表中。函数需要通过调用先前开发的load\_doc（）函数来加载文档。它需要使用先前开发的clean\_doc（）函数清理加载的文档，然后需要将所有标记添加到计数器，并更新计数。我们可以通过调用counter对象上的update（）函数来完成最后一步。

Below is a function called add\_doc\_to\_vocab() that takes as arguments a document filename and a Counter vocabulary.  
下面是一个名为add\_doc\_to\_vocab（）的函数，它以文档文件名和计数器词汇表作为参数。

|  |  |
| --- | --- |
| 12345678 一千二百三十四万五千六百七十八 | # load doc and add to vocabdef add\_doc\_to\_vocab(filename, vocab): # load doc doc = load\_doc(filename) # clean doc tokens = clean\_doc(doc) # update counts vocab.update(tokens) #加载doc并添加到vocabdef add\_doc\_to\_vocab（filename，vocab）：35;加载doc doc=load\_doc（filename）#清除doc tokens=clean\_doc（doc）#更新计数vocab.update（tokens） |
|  |  |

Finally, we can use our template above for processing all documents in a directory called process\_docs() and update it to call add\_doc\_to\_vocab().  
最后，我们可以使用上面的模板处理名为process\_docs（）的目录中的所有文档，并将其更新为调用add\_doc\_to\_vocab（）。

|  |  |
| --- | --- |
| 1234567891011 一兆二千三百四十五亿六千七百八十九万一千零一十一 | # load all docs in a directorydef process\_docs(directory, vocab): # walk through all files in the folder for filename in listdir(directory): # skip files that do not have the right extension if not filename.endswith(“.txt”): continue # create the full path of the file to open path = directory + ‘/’ + filename # add doc to vocab add\_doc\_to\_vocab(path, vocab) #加载directorydef进程中的所有文档（目录，vocab）：#遍历listdir（directory）中filename文件夹中的所有文件：#如果不是filename，则跳过没有正确扩展名的文件。endswith（“.txt”）：继续#创建要打开的文件的完整路径path=directory+'/'+filename#将doc添加到vocab将doc添加到vocab（path，vocab） |
|  |  |

We can put all of this together and develop a full vocabulary from all documents in the dataset.  
我们可以将所有这些放在一起，并从数据集中的所有文档中开发完整的词汇表。

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061 12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061 | from string import punctuationfrom os import listdirfrom collections import Counterfrom nltk.corpus import stopwords # load doc into memorydef load\_doc(filename): # open the file as read only file = open(filename, ‘r’) # read all text text = file.read() # close the file file.close() return text # turn a doc into clean tokensdef clean\_doc(doc): # split into tokens by white space tokens = doc.split() # remove punctuation from each token table = str.maketrans(’‘,’‘, punctuation) tokens = [w.translate(table) for w in tokens] # remove remaining tokens that are not alphabetic tokens = [word for word in tokens if word.isalpha()] # filter out stop words stop\_words = set(stopwords.words(’english’)) tokens = [w for w in tokens if not w in stop\_words] # filter out short tokens tokens = [word for word in tokens if len(word) > 1] return tokens # load doc and add to vocabdef add\_doc\_to\_vocab(filename, vocab): # load doc doc = load\_doc(filename) # clean doc tokens = clean\_doc(doc) # update counts vocab.update(tokens) # load all docs in a directorydef process\_docs(directory, vocab): # walk through all files in the folder for filename in listdir(directory): # skip files that do not have the right extension if not filename.endswith(“.txt”): continue # create the full path of the file to open path = directory + ‘/’ + filename # add doc to vocab add\_doc\_to\_vocab(path, vocab) # define vocabvocab = Counter()# add all docs to vocabprocess\_docs(‘txt\_sentoken/neg’, vocab)process\_docs(‘txt\_sentoken/pos’, vocab)# print the size of the vocabprint(len(vocab))# print the top words in the vocabprint(vocab.most\_common(50)) 从字符串导入标点符号从操作系统导入列表dirfrom collections import Counterfrom nltk.corpus import stopwords#将文档加载到memorydef loaddoc（filename）：ť将文件打开为只读文件=打开（filename，'r'）ť读取所有文本=文件。read（）ť关闭文件。close（）返回文本ť将文档转换为干净的标记df cleandoc（doc）：#按空白标记拆分为标记=doc.split（）#从每个标记表中删除标点符号=str.maketrans（''，标点符号）标记=[w.translate（table）for w in tokens]#删除不按字母顺序排列的标记=[word for word in tokens if word.isalpha（）]ţ筛选出停止词stopţwords=set（stopwords.words（'english'）标记=[w for w in tokens if not w in stopţwords]ţ筛选出短标记标记标记标记=[word in tokens iflen（word）>1]return tokens#load doc and add to vocadef add \_doc \_vocab（filename，vocab）：#load doc doc=load \_doc（filename）#clean doc tokens=clean \_doc（doc）ţupdate counts vocab.update（tokens）ţ加载directorydef进程\_docs（目录，vocab）：#遍历listdir（directory）中filename文件夹中的所有文件：#如果不是filename，则跳过没有正确扩展名的文件。endswith（“.txt”）：继续#创建要打开的文件的完整路径path=directory+'/'+filename#add doc to vocab add doc~ to vocab（path，vocab）#define vocabvcanble=Counter（）#将所有文档添加到vocab process\_docs（'txt\_sentoken/neg'，vocab）process\_docs（'txt\_sentoken/pos'，vocab）#打印出vocab print的大小（len（vocab））#打印出vocabprint中的最上面的单词（vocab.mostúcommon（50）） |
|  |  |

Running the example creates a vocabulary with all documents in the dataset, including positive and negative reviews.  
运行该示例将创建一个词汇表，其中包含数据集中的所有文档，包括正面和负面评论。

We can see that there are a little over 46,000 unique words across all reviews and the top 3 words are ‘film‘, ‘one‘, and ‘movie‘.  
我们可以看到，在所有的评论中有超过46000个独特的单词，前三个单词是“film”、“one”和“movie”。

|  |  |
| --- | --- |
| 12 德意志北方银行 | 46557[(‘film’, 8860), (‘one’, 5521), (‘movie’, 5440), (‘like’, 3553), (‘even’, 2555), (‘good’, 2320), (‘time’, 2283), (‘story’, 2118), (‘films’, 2102), (‘would’, 2042), (‘much’, 2024), (‘also’, 1965), (‘characters’, 1947), (‘get’, 1921), (‘character’, 1906), (‘two’, 1825), (‘first’, 1768), (‘see’, 1730), (‘well’, 1694), (‘way’, 1668), (‘make’, 1590), (‘really’, 1563), (‘little’, 1491), (‘life’, 1472), (‘plot’, 1451), (‘people’, 1420), (‘movies’, 1416), (‘could’, 1395), (‘bad’, 1374), (‘scene’, 1373), (‘never’, 1364), (‘best’, 1301), (‘new’, 1277), (‘many’, 1268), (‘doesnt’, 1267), (‘man’, 1266), (‘scenes’, 1265), (‘dont’, 1210), (‘know’, 1207), (‘hes’, 1150), (‘great’, 1141), (‘another’, 1111), (‘love’, 1089), (‘action’, 1078), (‘go’, 1075), (‘us’, 1065), (‘director’, 1056), (‘something’, 1048), (‘end’, 1047), (‘still’, 1038)] |
|  |  |

Perhaps the least common words, those that only appear once across all reviews, are not predictive. Perhaps some of the most common words are not useful too.  
也许最不常见的词，那些只在所有评论中出现一次的词，并不是预测性的。也许一些最常见的词也没用。

These are good questions and really should be tested with a specific predictive model.  
这些都是好问题，真的应该用一个特定的预测模型来测试。

Generally, words that only appear once or a few times across 2,000 reviews are probably not predictive and can be removed from the vocabulary, greatly cutting down on the tokens we need to model.  
一般来说，在2000篇评论中只出现一次或几次的单词可能不具有预测性，可以从词汇表中删除，从而大大减少我们需要建模的标记。

We can do this by stepping through words and their counts and only keeping those with a count above a chosen threshold. Here we will use 5 occurrences.  
我们可以通过单步执行单词和它们的计数，并且只保留那些计数高于选定阈值的单词。这里我们将使用5次。

|  |  |
| --- | --- |
| 1234 | # keep tokens with > 5 occurrencemin\_occurane = 5tokens = [k for k,c in vocab.items() if c >= min\_occurane]print(len(tokens)) |
|  |  |

This reduces the vocabulary from 46,557 to 14,803 words, a huge drop. Perhaps a minimum of 5 occurrences is too aggressive; you can experiment with different values.  
这使得词汇量从46557个单词减少到14803个单词，降幅很大。可能至少出现5次过于激进；您可以尝试使用不同的值。

We can then save the chosen vocabulary of words to a new file. I like to save the vocabulary as ASCII with one word per line.  
然后我们可以将所选词汇表保存到一个新文件中。我喜欢将词汇表保存为每行一个单词的ASCII。

Below defines a function called save\_list() to save a list of items, in this case, tokens to file, one per line.  
下面定义了一个名为save\_list（）的函数来保存项列表，在本例中，标记保存到文件中，每行一个。

|  |  |
| --- | --- |
| 12345 | def save\_list(lines, filename): data = ‘’.join(lines) file = open(filename, ‘w’) file.write(data) file.close() |
|  |  |

The complete example for defining and saving the vocabulary is listed below.  
下面列出了定义和保存词汇表的完整示例。

from string import punctuationfrom os import listdirfrom collections import Counterfrom nltk.corpus import stopwords # load doc into memorydef load\_doc(filename): # open the file as read only file = open(filename, 'r') # read all text text = file.read() # close the file file.close() return text # turn a doc into clean tokensdef clean\_doc(doc): # split into tokens by white space tokens = doc.split() # remove punctuation from each token table = str.maketrans('', '', punctuation) tokens = [w.translate(table) for w in tokens] # remove remaining tokens that are not alphabetic tokens = [word for word in tokens if word.isalpha()] # filter out stop words stop\_words = set(stopwords.words('english')) tokens = [w for w in tokens if not w in stop\_words] # filter out short tokens tokens = [word for word in tokens if len(word) > 1] return tokens # load doc and add to vocabdef add\_doc\_to\_vocab(filename, vocab): # load doc doc = load\_doc(filename) # clean doc tokens = clean\_doc(doc) # update counts vocab.update(tokens) # load all docs in a directorydef process\_docs(directory, vocab): # walk through all files in the folder for filename in listdir(directory): # skip files that do not have the right extension if not filename.endswith(".txt"): continue # create the full path of the file to open path = directory + '/' + filename # add doc to vocab add\_doc\_to\_vocab(path, vocab) # save list to filedef save\_list(lines, filename): data = '\n'.join(lines) file = open(filename, 'w') file.write(data) file.close() # define vocabvocab = Counter()# add all docs to vocabprocess\_docs('txt\_sentoken/neg', vocab)process\_docs('txt\_sentoken/pos', vocab)# print the size of the vocabprint(len(vocab))# print the top words in the vocabprint(vocab.most\_common(50))# keep tokens with > 5 occurrencemin\_occurane = 5tokens = [k for k,c in vocab.items() if c >= min\_occurane]print(len(tokens))# save tokens to a vocabulary filesave\_list(tokens, 'vocab.txt')  
从字符串导入标点符号从操作系统导入列表目录从集合导入反存根nltk.corpus导入stopwords#将文档加载到memorydef加载文档（文件名）：#将文件作为只读文件打开（文件名，'r'）#读取所有文本文本=文件.read（）#关闭文件.close（）返回文本#将文档转换为干净的标记def cleanŠu doc（doc）：Š按空格标记拆分为标记=doc.split（）Š从每个标记表中删除标点符号=str.maketrans（''，标点符号）标记=[w.translate（table）for w in tokens]#删除不按字母顺序排列的标记=[word for word in tokens if word.isalpha（）]ţ筛选出停止词stopţwords=set（stopwords.words（'english'）标记=[w for w in tokens if not w in stopţwords]ţ筛选出短标记标记标记标记=[word for word in如果len（word）>1]返回tokens#加载doc并添加到vocabdef add \_doc \_vocab（filename，vocab）：ţ加载doc doc=load \_doc（filename）ţ清除doc tokens=clean \_doc（doc）ţ更新计数vocab.update（tokens）ţ加载directorydef进程\_docs（目录，vocab）：#遍历listdir（directory）中filename文件夹中的所有文件：#如果不是filename，则跳过没有正确扩展名的文件。endswith（“.txt”）：继续#创建要打开的文件的完整路径path=directory+'/'+filename#add doc to vocab addúdocútoúvocab（path，vocab）#将列表保存到文件定义保存列表（行，文件名）：data='\n'。join（行）file=open（文件名，'w'）file.write（数据）file.close（）；define vocabvcanc=Counter（）；将所有文档添加到vocab process\u docs（'txt\_sentoken/neg'，vocab）process\u docs（'txt\_sentoken/pos'，vocab）#打印vocabprint的大小（len（vocab））#打印vocabprint中的最上面的单词（vocab.most戋common（50））#保留出现次数大于5的标记min戋ochrane=5tokens=[k代表k，c代表vocab.items（），如果c>=min戋ochrane]打印（len（tokens））#将标记保存到词汇文件保存列表（tokens，'vocab.txt'）

Running this final snippet after creating the vocabulary will save the chosen words to file.  
创建词汇表后运行最后一段代码将把所选单词保存到文件中。

It is a good idea to take a look at, and even study, your chosen vocabulary in order to get ideas for better preparing this data, or text data in the future.  
这是一个好主意，看看，甚至研究，你选择的词汇，以便获得更好地准备这些数据，或文本数据在未来的想法。

python hasntupdatingfigurativelysymphonyciviliansmightfishermanhokumwitchbuffoons...  
巨蟒已经更新了象征性的符号生命和渔夫巫术小丑。。。

Next, we can look at using the vocabulary to create a prepared version of the movie review dataset.  
接下来，我们可以看看如何使用词汇表创建电影评论数据集的准备版本。

## 5. Save Prepared Data 5个。保存准备好的数据

We can use the data cleaning and chosen vocabulary to prepare each movie review and save the prepared versions of the reviews ready for modeling.  
我们可以使用数据清理和选定的词汇表来准备每个电影评论，并保存准备好的评论版本以备建模。

This is a good practice as it decouples the data preparation from modeling, allowing you to focus on modeling and circle back to data prep if you have new ideas.  
这是一个很好的实践，因为它将数据准备与建模分离开来，允许您专注于建模，如果您有新的想法，则返回到数据准备。

We can start off by loading the vocabulary from ‘vocab.txt‘.  
我们可以从“vocab.txt”加载词汇表开始。

# load doc into memory  
def load\_doc(filename):   
 # open the file as read only   
 file = open(filename, 'r')   
 # read all text   
 text = file.read()   
 # close the file file.close()   
 return text # load vocabulary  
vocab\_filename = 'review\_polarity/vocab.txt'  
vocab = load\_doc(vocab\_filename)  
vocab = vocab.split()  
vocab = set(vocab)  
#将文档加载到memorydef加载文档（文件名）：#以只读文件的形式打开文件=打开（文件名，'r'）#读取所有文本文本=文件.读取（）#关闭文件.关闭（）返回文本#加载词汇词汇词汇词汇文件名='查看极性/vocab.txt'vocab=加载文档（词汇文件名）vocab=vocab.split（）vocab=set（词汇）

Next, we can clean the reviews, use the loaded vocab to filter out unwanted tokens, and save the clean reviews in a new file.  
接下来，我们可以清理评论，使用加载的vocab过滤掉不需要的标记，并将干净的评论保存到一个新文件中。

One approach could be to save all the positive reviews in one file and all the negative reviews in another file, with the filtered tokens separated by white space for each review on separate lines.  
一种方法是将所有的正面评论保存在一个文件中，而将所有的负面评论保存在另一个文件中，过滤后的标记用空格分隔，每个评论在单独的行上。

First, we can define a function to process a document, clean it, filter it, and return it as a single line that could be saved in a file. Below defines the doc\_to\_line() function to do just that, taking a filename and vocabulary (as a set) as arguments.  
首先，我们可以定义一个函数来处理一个文档，清除它，过滤它，并将它作为一行返回，该行可以保存在一个文件中。下面定义了doc\_to\_line（）函数来实现这一点，将文件名和词汇表（作为一个集合）作为参数。

It calls the previously defined load\_doc() function to load the document and clean\_doc() to tokenize the document.  
它调用前面定义的load\_doc（）函数来加载文档，并调用clean\_doc（）来标记文档。

# load doc, clean and return line of tokens  
def doc\_to\_line(filename, vocab):   
 # load the doc   
 doc = load\_doc(filename)   
 # clean doc   
 tokens = clean\_doc(doc)   
 # filter by vocab   
 tokens = [w for w in tokens if w in vocab]  
 return ' '.join(tokens)  
#加载doc，clean并返回doc的标记行def docŠto\_line（filename，vocab）：Š加载doc doc=loadŠdoc（filename）Šclean doc tokens=cleanŠdoc（doc）Š按vocab tokens筛选=[w for w in tokens if w in vocab]返回'。join（tokens）

Next, we can define a new version of process\_docs() to step through all reviews in a folder and convert them to lines by calling doc\_to\_line() for each document. A list of lines is then returned.  
接下来，我们可以定义一个新版本的process\_docs（）来遍历文件夹中的所有评论，并通过为每个文档调用doc\_to\_line（）将它们转换为行。然后返回一个行列表。

# load all docs in a directory  
def process\_docs(directory, vocab):   
 lines = list()   
 # walk through all files in the folder   
 for filename in listdir(directory):   
 # skip files that do not have the right extension   
 if not filename.endswith(".txt"):   
 continue   
 # create the full path of the file to open   
 path = directory + '/' + filename   
 # load and clean the doc   
 line = doc\_to\_line(path, vocab)   
 # add to list   
 lines.append(line)   
 return lines  
#加载directorydef进程中的所有文档\_docs（directory，vocab）：lines=list（）#遍历listdir（directory）中filename文件夹中的所有文件：#如果不是filename，则跳过没有正确扩展名的文件。endswith（“.txt”）：继续#创建完整路径打开文件的路径=目录+'/'+文件名#加载并清除doc line=doc\_to\_line（path，vocab）#添加到列表行。追加（line）返回行

We can then call process\_docs() for both the directories of positive and negative reviews, then call save\_list() from the previous section to save each list of processed reviews to a file.  
然后，我们可以为正面评论和负面评论的目录调用process\_docs（），然后从上一节调用save\_list（）将每个已处理评论的列表保存到一个文件中。

The complete code listing is provided below.  
下面提供了完整的代码列表。

from string import punctuation  
from os import listdir  
from collections import Counter  
from nltk.corpus import stopwords   
# load doc into memory  
def load\_doc(filename):   
 # open the file as read only   
 file = open(filename, 'r')   
 # read all text text = file.read()   
 # close the file   
 file.close()   
 return text   
# turn a doc into clean tokens  
def clean\_doc(doc):   
 # split into tokens by white space   
 tokens = doc.split()   
 # remove punctuation from each token   
 table = str.maketrans('', '', punctuation)   
 tokens = [w.translate(table) for w in tokens]   
 # remove remaining tokens that are not alphabetic   
 tokens = [word for word in tokens if word.isalpha()]   
 # filter out stop words   
 stop\_words = set(stopwords.words('english'))   
 tokens = [w for w in tokens if not w in stop\_words]   
 # filter out short tokens   
 tokens = [word for word in tokens if len(word) > 1]   
 return tokens   
# save list to file  
def save\_list(lines, filename):   
 data = '\n'.join(lines)   
 file = open(filename, 'w')   
 file.write(data)   
 file.close()   
# load doc, clean and return line of tokens  
def doc\_to\_line(filename, vocab):   
 # load the doc   
 doc = load\_doc(filename)   
 # clean doc   
 tokens = clean\_doc(doc)   
 # filter by vocab   
 tokens = [w for w in tokens if w in vocab]   
 return ' '.join(tokens)   
# load all docs in a directory  
def process\_docs(directory, vocab):   
 lines = list()   
 # walk through all files in the folder   
 for filename in listdir(directory):   
 # skip files that do not have the right extension   
 if not filename.endswith(".txt"):   
 continue   
 # create the full path of the file to open path = directory + '/' + filename   
 # load and clean the doc   
 line = doc\_to\_line(path, vocab)   
 # add to list   
 lines.append(line)   
 return lines   
# load vocabulary  
vocab\_filename = 'vocab.txt'  
vocab = load\_doc(vocab\_filename)  
vocab = vocab.split()  
vocab = set(vocab)  
# prepare negative reviews  
negative\_lines = process\_docs('txt\_sentoken/neg', vocab)  
save\_list(negative\_lines, 'negative.txt')  
# prepare positive reviews  
positive\_lines = process\_docs('txt\_sentoken/pos', vocab)  
save\_list(positive\_lines, 'positive.txt')  
从字符串导入标点符号从操作系统导入列表目录从集合导入反存根nltk.corpus导入stopwords#将文档加载到memorydef load#doc（filename）：#将文件打开为只读文件=打开（filename，'r'）#读取所有文本=文件。read（）#关闭文件。close（）返回文本#将文档转换为clean tokensdef clean\_doc（doc）：#按空格tokens=doc.split（）拆分为标记#从每个标记表中删除标点符号=str.maketrans（''，标点符号）标记=[w.translate（table）for w in tokens]#删除其余非字母标记的标记=[word for word in tokens if word.isalpha（）]ţ筛选出停止词stopţwords=set（stopwords.words（'english'）标记=[w for w in tokens if not w in stopţwords]ţ筛选出短标记tokens=[len（word）>1]返回tokens#save list to filedef save\_list（lines，filename）：data='\n'。join（lines）file=open（filename，'w'）file.write（data）file.close（）#加载doc，clean，并将tokens的行sdef doc返回到line（filename，vocab）：#加载doc doc=load\_doc（filename）#clean doc tokens=clean\_doc（doc）#按vocab tokens筛选=[w for w in tokens if w in vocab]return'。join（tokens）#加载directorydef进程中的所有文档，vocab）：lines=list（）#遍历listdir（directory）中filename文件夹中的所有文件：#如果不是filename，则跳过没有正确扩展名的文件。endswith（“.txt”）：继续#创建文件的完整路径以打开path=directory+'/'+filename#加载并清理doc line=doc\_to\_line（路径，vocab）#添加到列表行。追加（行）返回行ţ加载vocabularyvocabŧ文件名='vocab.txt'vocab=加载文档（vocabŧ文件名）vocab=vocab.split（）vocab=set（vocab）ŧ准备负的reviewsnegative lines=process docs（'txt sentoken/neg'，vocab）保存列表（负的lines，'negative.txt'）#准备正的reviewspositive\_lines=process\_docs（'txt\_sentoken/pos'，vocab）保存列表（positive\_lines，'positive.txt'）

Running the example saves two new files, ‘negative.txt‘ and ‘positive.txt‘, that contain the prepared negative and positive reviews respectively.  
运行此示例将保存两个新文件“negative.txt”和“positive.txt”，它们分别包含准备好的负面评论和正面评论。

The data is ready for use in a bag-of-words or even word embedding model.  
这些数据可以在一个单词包甚至单词嵌入模型中使用。

## Extensions 扩展

This section lists some extensions that you may wish to explore.  
本节列出了一些您可能希望探索的扩展。

* Stemming. We could reduce each word in documents to their stem using a stemming algorithm like the Porter stemmer.  
  堵塞。我们可以使用像Porter词干分析器这样的词干分析算法将文档中的每个单词都减少到它们的词干。
* N-Grams. Instead of working with individual words, we could work with a vocabulary of word pairs, called bigrams. We could also investigate the use of larger groups, such as triplets (trigrams) and more (n-grams).  
  N克。我们不需要处理单个单词，而是可以处理一个单词对的词汇表，称为bigrams。我们还可以研究使用更大的组，如三胞胎（trigrams）和更多（n-grams）。
* Encode Words. Instead of saving tokens as-is, we could save the integer encoding of the words, where the index of the word in the vocabulary represents a unique integer number for the word. This will make it easier to work with the data when modeling.  
  编码单词。与按原样保存标记不同，我们可以保存单词的整数编码，其中词汇表中单词的索引表示单词的唯一整数。这将使建模时更容易使用数据。
* Encode Documents. Instead of saving tokens in documents, we could encode the documents using a bag-of-words model and encode each word as a boolean present/absent flag or use more sophisticated scoring, such as TF-IDF.  
  对文档进行编码。与在文档中保存标记不同，我们可以使用一个单词袋模型对文档进行编码，并将每个单词编码为布尔的present/absent标志，或者使用更复杂的评分，如TF-IDF。

If you try any of these extensions, I’d love to know. Share your results in the comments below.  
如果你尝试这些扩展，我很想知道。在下面的评论中分享你的结果。

## Further Reading 进一步阅读

This section provides more resources on the topic if you are looking go deeper.  
如果您想深入了解，本节将提供更多有关此主题的资源。

### Dataset

* [Movie Review Data](http://www.cs.cornell.edu/people/pabo/movie-review-data/)
* [A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts](http://xxx.lanl.gov/abs/cs/0409058), 2004.
* [Movie Review Polarity Dataset](http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz) (.tgz)
* Dataset Readme [v2.0](http://www.cs.cornell.edu/people/pabo/movie-review-data/poldata.README.2.0.txt) and [v1.1](http://www.cs.cornell.edu/people/pabo/movie-review-data/README.1.1).

### APIs

* [nltk.tokenize package API](http://www.nltk.org/api/nltk.tokenize.html)
* [Chapter 2, Accessing Text Corpora and Lexical Resources](http://www.nltk.org/book/ch02.html)
* [os API Miscellaneous operating system interfaces](https://docs.python.org/3/library/os.html)
* [collections API – Container datatypes](https://docs.python.org/3/library/collections.html)

## Summary

In this tutorial, you discovered how to prepare movie review text data for sentiment analysis, step-by-step.

Specifically, you learned:

* How to load text data and clean it to remove punctuation and other non-words.
* How to develop a vocabulary, tailor it, and save it to file.
* How to prepare movie reviews using cleaning and a predefined vocabulary and save them to new files ready for modeling.

Do you have any questions? Ask your questions in the comments below and I will do my best to answer.