# 第9章 **Python中的社交媒体挖掘**

这一章我们要来讨论一下社交媒体。虽然这方面的内容与NLTK/NLP没有直接的关系，但社交数据也是一种非常丰富的非结构化文本的数据源。所以，作为NLP爱好者，我们应该掌握一些摆弄社交数据的技能。在本章，我们将会探讨如何试着从一些目前最受欢迎的社交媒体平台中收集相关数据。我们将会介绍如何利用一些Python API来从Twitter、Facebook等社交媒体中收集数据。我们还会探讨一些在社交媒体挖掘领域中最常见的用例，例如热门话题、情绪分析等。

我们在前面的章节中已经学习了许多与自然语言处理和机器学习相关的概念性话题。在本章，我们将会试着围绕着一些社交数据来构建一些应用程序。我们还将在这里提供一些针对社交数据处理的最佳实践，并带您以可视化图形的方式来查看这些社交数据。

社交媒体中都会存在一个基础性的图结构，而大多数基于图结构的问题都可以被表述成某种信息流问题，并找出该图结构中出入最繁忙的节点。像热门话题、影响力检测以及情绪分析这些问题都是很好的例子。下面，我们就来通过这些具体的用例，围绕着社交网络来构建一些酷炫的应用程序吧。

在阅读完本章之后，我们希望：

* 您应该知道如何用相关API收集任意社交媒体中的数据。
* 您也应该学会了如何用某种结构化格式来表述数据，并以此构建出一些很棒的应用程序。
* 最后，您应该能为社交数据绘制可视化图形，并能对其进行有意义的观察。

## 数据收集

本章最重要的目标是要介绍如何在一些业界最常见的社交网络之间进行数据收集。我们这里将主要以Twitter和Facebook为实验对象，我们会为您详细、充分地介绍与这两个社交媒体有关的API信息，以及如何有效地利用它们来获取相关数据。此外，我们还将与废弃数据相关的数据字典，以及如何利用这些我们目前所学到的知识来够闲一些酷炫的应用程序。

### Twitter

我们先从一个目前最流行、最开放的、且完全公开的社交媒体开始入手。这实际上就意味着我们可能要去收集整个Twitter流中的信息，但这是要付费的，然而我们可以免费捕获其中百分之一的信息。在商业背景下，对于那些想要了解公众情绪、新兴话题这类信息的人来说，Twitter可是一个非常丰富的信息资源。

下面，我们就来面对如何从tweets中获取与用例相关信息这一主要挑战吧。

|  |
| --- |
| 下面链接中列出了许多Twitter程序库的代码仓库[[1]](#footnote-1)。当然，这些程序库都没有经过Twitter官方的验证，但它们都可以基于Twitter API来运行。  <https://dev.twitter.com/overview/api/twitterlibraries> |

具有这方面功能的Python库绝对有超过10个以上，所以我们可以任意选择一个自己自己喜欢的。由于我自己通常会选择Tweepy，所以这里将会使用它来完成本书中的示例。 这些程序库大部分都是一些Twitter API的封装器，因此它们的参数和签名也都大致相同。

安装Tweepy最简单的方法是使用pip：

$ pip install tweepy

|  |
| --- |
| 源代码安装是比较难的一种安装方式，Tweepy在github上的链接如下：  <https://github.com/tweepy/tweepy> 。 |

当然，如果想要让Tweepy能够正常工作，我们还必须在Twitter上创建一个开发者帐户，为将要创建的的应用获取访问令牌。在完成这些动作之后，我们就会得到属于自己的的资格验证以及这些验证信息下面的密钥。您可以在<https://apps.twitter.com/app/new>中注册并获取令牌。下图显示的就是一个访问令牌的快照：

（图）

我们先来看一个非常简单的例子：通过Twitter信息流的API来收集数据。我们会用Tweepy来捕获Twitter流，收集其中所有与给定关键字相关的tweets：

tweetdump.py

>>> from tweepy.streaming import StreamListener

>>> from tweepy import OAuthHandler

>>> from tweepy import Stream

>>> import sys

>>> consumer\_key = 'ABCD012XXXXXXXXx'

>>> consumer\_secret = 'xyz123xxxxxxxxxxxxx'

>>> access\_token = '000000-ABCDXXXXXXXXXXX'

>>> access\_token\_secret ='XXXXXXXXXgaw2KYz0VcqCO0F3U4'

>>> class StdOutListener(StreamListener):

>>> def on\_data(self, data):

>>> with open(sys.argv[1],'a') as tf:

>>> tf.write(data)

>>> return

>>> def on\_error(self, status):

>>> print(status) >>>if \_\_name\_\_ == '\_\_main\_\_':

>>> l = StdOutListener()

>>> auth = OAuthHandler(consumer\_key, consumer\_secret)

>>> auth.set\_access\_token(access\_token, access\_token\_secret)

>>> stream = Stream(auth, l)

>>> stream.filter(track=['Apple watch'])

在上述代码中，我们使用了与Tweepy示例相同的代码，并稍许作了些修改。这个例子示范了如何使用Twitter信息流的API，我们跟踪的关键词是**Apple Watch**。Twitter信息流的API在这里实际提供的就是在Twitter信息流中执行搜索的功能，我们可以用该API查看其信息流中最多百分之一的信息。

对于上述代码，我们主要需要理解的部分是头四行和最后四行。在初始化的那几行代码中，我们指定的是上一节中生成的访问令牌和其它相关密钥。而在最后四行中，我们创建了一个针对信息流的监听器。特别在最后一行，我们使用了stream.filter来过滤twitter，以便设置我们要跟踪的关键字。我们在这里可以一次设置多个关键字。在这个例子中，我运行结果中将包含所有与Apple Watch这个词相关的tweets。

在下面的示例中，我们会将上面所收集的tweets载入进来，带您来看看tweet的结构，并探讨如何从中提取出有意义的信息。通常情况下，tweet JSON在结构上应该是这样的：

{

"created\_at":"Wed May 13 04:51:24 +0000 2015",

"id":598349803924369408,

"id\_str":"598349803924369408",

"text":"Google launches its first Apple Watch app with News & Weather http:\/\/t.co\/o1XMBmhnH2",

"source":"\u003ca href=\"http:\/\/ifttt.com\" rel=\"nofollow\"\ u003eIFTTT\u003c\/a\u003e", "truncated":false,

"in\_reply\_to\_status\_id":null,

"user":{

"id":1461337266,

"id\_str":"1461337266",

"name":"vestihitech \u0430\u0432\u0442\u043e\u043c\u0430\u0442",

"screen\_name":"vestihitecha",

"location":"",

"followers\_count":20,

"friends\_count":1,

"listed\_count":4,

""statuses\_count":7442,

"created\_at":"Mon May 27 05:51:27 +0000 2013",

"utc\_offset":14400,

},

,

"geo":{ "latitude" : 51.4514285, "longitude"=-0.99

}

"place":"Reading, UK",

"contributors":null,

"retweet\_count":0,

"favorite\_count":0,

"entities":{

"hashtags":["apple watch", "google"

],

"trends":[

],

"urls":[

{

"url":"http:\/\/t.co\/o1XMBmhnH2",

"expanded\_url":"http:\/\/ift.tt\/1HfqhCe",

"display\_url":"ift.tt\/1HfqhCe",

"indices":[

66,

88

]

}

],

"user\_mentions":[

],

"symbols":[

]

},

"favorited":false,

"retweeted":false,

"possibly\_sensitive":false,

"filter\_level":"low",

"lang":"en",

"timestamp\_ms":"1431492684714"

}

]

## 数据提取

在数据提取的过程中，我们有一些最常用到的字段：

* text：即用户所提供的tweet内容。
* user：即用户的一些主要属性，譬如用户名、所在地和照片等
* Place：即发布tweet的地方，也是地理坐标。
* Entities：即用户在其tweet中所附加的有效主题标签和主题

在实践中，上面所提到的每个属性都可以成为执行社交媒体挖掘很好的实验用例。下面，我们要来看看应该如何获取这些属性，并将其转换成某种可读性更好的形式，或者对其进其它相关的处理：

Source: tweetinfo.py

>>> import json

>>> import sys

>>> tweets = json.loads(open(sys.argv[1]).read())

>>> tweet\_texts = [ tweet['text']\

for tweet in tweets ]

>> >tweet\_source = [tweet ['source'] for tweet in tweets]

>>> tweet\_geo = [tweet['geo'] for tweet in tweets]

>>> tweet\_locations = [tweet['place'] for tweet in tweets]

>>> hashtags = [ hashtag['text'] for tweet in tweets for hashtag in

tweet['entities']['hashtags'] ]

>>> print tweet\_texts

>>> print tweet\_locations

>>> print tweet\_geo

>>> print hashtags

正如我们所期待的那样，上述代码会输出四个列表，其中，tweet\_texts中所包含的是所有的tweet内容，此外还有该tweet所在的位置和主题标签。

|  |
| --- |
| 在这段代码中，我们只是用json.loads()加载了一段以JSON格式生成的输出。我会建议读者使用像Json Parser（http://json.parser.online.fr/）这样的线上工具来帮助您理解这段JSON输出的含义，以及它的属性（键和值）内容。 |

接下来，如果我们仔细观察的话，会发现这段JSON被分成了不同的层次，其中有些属性，譬如说text就只有一个直接值，而有些属性则更多的是一些嵌套信息。这就是为什么我们会看到如果想要查看hashtag属性，就必须迭代上几个层次，而对于test属性，我们就只要直接获取值即可。由于我们的文件实际是一个tweets列表，所以必须要通过迭代该列表才能获取到所有的tweets，而这里每个tweet对象的结构和上面那个tweet示例是基本相同的。

### 热门话题

现在，如果我们想要这种设置条件下查看查找出当前热门的话题。最简单一种方法就是查看所有tweet中单词的频率分布情况。因为我们目前已经有了一个包含所有tweet的tweet\_text列表：

>>> import nltk

>>> from nltk import word\_tokenize,sent\_tokenize

>>> from nltk import FreqDist

>>> tweets\_tokens = []

>>> for tweet in tweet\_text:

>>> tweets\_tokens.append(word\_tokenize(tweet))

>>> Topic\_distribution = nltk.FreqDist(tweets\_tokens)

>>> Freq\_dist\_nltk.plot(50, cumulative=False)

除此之外，我们还有一个更复杂一些的方法，就是利用我们*在第3章“词性标注”*中所学到的词性标注器。其原理是：在大多数情况下，我们的主题都是一些名词或实体项。所以，我们可以对其执行相同的处理。在上面的代码中，我们读取了每个tweet并对其进行了标识化，下面我们要以POS为过滤器，只选取其中的名词来充当主题：

>>> import nltk

>>> Topics = []

>>> for tweet in tweet\_text:

>>> tagged = nltk.pos\_tag(word\_tokenize(tweet))

>>> Topics\_token = [word for word,pos in ] in tagged if pos in ['NN','NNP']

>>> print Topics\_token

如果想让这个例子再酷炫一点，我们还可以跨时间收集一些tweet，然后为其生成绘制图。这样我们就会得到一份含义非常清楚的热门主报告。例如，我们之前正在寻找的是与 “Apple Watch”相关的数据。这个词应该会在苹果推出Apple Watch的那一天，以及该产品开始销售的那一天达到高峰。 但我们更感兴趣的是那几天除了这些主题，还出现了什么样的主题，这些主题随着时间呈现出了何种讨论趋势。

## 地理可视化

地理位置的可视化也是社交媒体的常见应用之一。 在tweet结构中，我们会看到geo、longitude和latitude这三个与之相关的属性。通过访问这些属性，我们就能利用像D3这样的可视化程序库做出下面这样的东西：

（图）

我们在这个例子中看到的只是其中一种可视化实现，它呈现的是全美tweet的可视化分布。我们从中可以清楚地看出纽约这些东部地区的使用强度正在增长。现在，一家公司可以通过对客户进行类似的分析来厘清其客户群喜欢呆在哪些地方。除此之外，我们还能根据情绪来对tweet进行文本挖掘，以此来推断哪几个州的客户对公司不太满意等情况。

### 影响力检测

在社交图语境中，对一个具有重要影响力的社交图中的重要节点进行检测也是一个很重要的问题。因此，如果我们手里有数百万条关于我们公司的tweet，那么其中一个重要用例就是要收集该社交媒体中最有影响力的那些客户，然后针对他们进行品牌推广、营销或改善他们的参与度。

如果具体到Twitter的情况，我们就得要回到图论和PageRank的概念上来，其中对于一个给定的节点，如果其出度（outdegree）比例比入度（indegree）高，该节点就是具有影响力的。原因非常直观，拥有较多关注者的人显然通常要比他们所关注的人有影响力。 有一家叫做KLOUT（https://klout.com/）的公司一直很关注这类问题。所以下面我就来写一个最基本而且直观的算法来为这些公司评个分：

>>>klout\_scores = [ (tweet['user']['followers\_count]/ tweet['user']

['friends\_count'],tweet['user']) for tweet in tweets ]

当然在这些以Twitter为基础的例子中，我们所修改的字段内容始终是完全相同。其实我们也可以用Facebook帖子来示范如何构建热门话题。同样地，我们也可以示范如何对Facebook用户、其所发帖的地理位置以及影响力进行可视化处理。 事实上，我们在下一节就要来看看这些应用在Facebook中会有哪些变化。

### Facebook

Facebookt的私人化程度要更高一些，有些接近于私人式的社交网络。Facebook出于隐私和安全方面的考虑，通常不会允许我们收集用户的订阅内容/帖子。因此，Facebook的图结构API只能以有限的方式来反馈给定页面，关于这方面的内容，我们推荐读者去阅读一下<https://developers.facebook.com/docs/graph-api/using-graph-api/v2.3>中的说明。以加深理解。

接下来要解决的一个问题是如何用Python来访问这些图结构API并加以应用。业界围绕着Facebook API编写了很多封装器，我们这里使用的是最常见的Facebook SDK：

$ pip install facebook-sdk

|  |
| --- |
| 我们也可以到以下链接中获取相应的安装包：  <https://github.com/Pythonforfacebook/facebook-sdk> |

然后，下一步就是要获取应用程序的访问令牌了。Facebook会将每个API调用都视为一个应用程序。因此即使是数据收集这一步骤，我们也需要将其伪装成一个应用程序。

|  |
| --- |
| 如果想要获得属于自己的访问令牌，请访问https://developers.facebook.com/tools/explorer. |

现在所有设置都完成了！下面我们先来试试一个使用度最高的Facebook的图结构API。在该API中，Facebook提供的是一个针对页面、用户、事件、地点等信息的基于图结构的搜索功能。这种情况下，获得相关帖子的过程变成了两个阶段，我们必须先找到与自己感兴趣的主题相关的pageid / userid，然后才能访问到该页面的订阅内容。下面我们就要来一个简单的实现用例，就是在某公司的官方页面上寻找客户投诉。其具体做法如下：

>>> import facebook

>>> import json

>>> fo = open("fdump.txt",'w')

>>> ACCESS\_TOKEN = 'XXXXXXXXXXX' # https://developers.facebook.com/tools/explorer

>>> fb = facebook.GraphAPI(ACCESS\_TOKEN)

>>> company\_page = "326249424068240"

>>> content = fb.get\_object(company\_page)

>>> fo.write(json.dumps(content))

这段代码要将访问令牌附加到Facebook的图结构API中，这样我们才能通过该API对Facebook做一个REST调用。但前提问题是我们必须要先获得给定页面的ID。下面我们来看看要附加的访问令牌，其代码如下：

"website":"www.dimennachildrenshistorymuseum.org",

"can\_post":true,

"category\_list":[

{

"id":"244600818962350",

"name":"History Museum"

},

{

"id":"187751327923426",

"name":"Educational Organization"

}

],

"likes":1793,

},

"id":"326249424068240",

"category":"Museum/art gallery",

"has\_added\_app":false,

"talking\_about\_count":8,

"location":{

"city":"New York",

"zip":"10024",

"country":"United States",

"longitude":-73.974413,

"state":"NY",

"street":"170 Central Park W",

"latitude":40.779236

},

"is\_community\_page":false,

"username":"nyhistorykids",

"description":"The first-ever museum bringing American history to life through the eyes of children, where kids plus history equals serious fun! Kids of all ages can practice their History Detective skills at the DiMenna Children's History Museum and:\n\n\u2022 discover the past through six historic figure pavilions\n\n\u2022!", "hours":{

""thu\_1\_close":"18:00"

},

"phone":"(212) 873-3400",

"link":"https://www.facebook.com/nyhistorykids",

"price\_range":"$ (0-10)",

"checkins":1011,

"about":"The DiMenna Children' History Museum is the first-ever museum bringing American history to life through the eyes of children. Visit it inside the New-York Historical Society!", "name":"New-York Historical Society DiMenna Children's History Museum",

"cover":{

"source":"https://scontent.xx.fbcdn.net/hphotos-xpf1/t31.0-8/s720x720/104

9166\_672951706064675\_339973295\_o.jpg",

"cover\_id":"672951706064675",

"offset\_x":0,

"offset\_y":54,

"id":"672951706064675"

},

"were\_here\_count":1011,

"is\_published":true

},

如您所见，这里所展现的Facebook数据模式与Twitter非常类似。接下来，我们可以来看看这个用例究竟请求到了怎样的信息。在大多数情况下，用户的post、category、name、about和likes都会是它的重要字段。当然，这个例子所演示的是一个博物馆的页面，而在更商业化的用例中，公司页面上应该会有一个长长的帖子列表和其他有用的信息，它们都可以为我们提供很好的观察点。

假设我们的组织xyz.org在Facebook上有一个专属页面，而我们现在想要了解有哪些用户在页面上投诉了我们。这对于投诉分类这样的应用来说也是一个很好的用例。目前实现这一应用的方式非常简单。我们需要先在fdump.txt中查找一组关键字，然后其操作的复杂程度就与在*第6章“文本分类”*中使用文本分类算法来进行评分差不多了。

现在我们来看另一个用例：查找感兴趣的主题，并在其结果页面中查看开放性帖子及其评论。这个功能与直接用Facebook主页上的图形搜索栏来搜索非常类似。但用编程的方式来做这件事的好处是我们可以自行主导这些搜索，并可以对每个页面上的用户评论进行递归搜索。下面来看看搜索用户数据的代码：

User search

>>> fb.request("search", {'q' : 'nitin', 'type' : 'user'}) Place based on the nearest location.

>>> fb.request("search", {'q' : 'starbucks', 'type' : 'place'}) Look for open pages.

>>> fb.request("search", {'q' : 'Stanford university', 'type' : page}) Look for event matching to the key word.

>>> fb.request("search", {'q' : 'beach party', 'type' : 'event'})

一旦我们将所有的相关数据都转换成了结构化格式，接下来就可以运用我们在探讨NLP和机器学习这些主题时所学到的那些概念了。下面我们选择用同一个用例来查找帖子，不过这回的主要找到是该Facebook页面上的投诉信息。

假设我们现在得到了以下格式的数据[[2]](#footnote-2)：

|  |  |
| --- | --- |
| **Userid** | **FB Post** |
| XXXX0001 | The product was pathetic and I tried reaching out to your customer care, but nobody responded |
| XXXX002 | Great work guys |
| XXXX003 | Where can I call to get my account activated ??? Really bad service |

在这里，我们等于是回到了在*第6章“文本分类”*中所用过的那个示例中，我们当时构建了一个文本分类器，并用它来检测某段SMS（消息文本）是否为垃圾邮件。类似地，我们在这里也可以利用来自于这些帖子集中这些数据来创建训练数据，在这里我们要这些手动标注器标注出那些带投诉内容的评论，其它则一概忽略。一旦我们得到了这些重要的训练数据，就可以构建出与相同的文本分类器了：

fb\_classification.py

>>> from sklearn.feature\_extraction.text import TfidfVectorizer

>>> vectorizer = TfidfVectorizer(min\_df=2, ngram\_range=(1, 2),

stop\_ words='english', strip\_accents='unicode', norm='l2')

>>> X\_train = vectorizer.fit\_transform(x\_train)

>>> X\_test = vectorizer.transform(x\_test)

>>> from sklearn.linear\_model import SGDClassifier

>>> clf = SGDClassifier(alpha=.0001, n\_iter=50).fit(X\_train, y\_train)

>>> y\_pred = clf.predict(X\_test)

接下来，假设这里只有三种样本。我们可以将其中的第一种和第三种标注为投诉类信息，第二种则是非投诉类信息。尽管我们好像是以相同的方式构建了一个单元或二元模型的向量化器，但事实上我们在这里是可以用相同的处理过程来构建一个分类器的。因为我们忽略了这里的一些预处理步骤。您可以在这里执行与*第6章“文本分类”*中相同的处理过程。当然在某些情况下，想要得到像上面这样获得训练数据是非常困难或者代价高昂的。所以对于其中的一些情况，我们也可以采用像文本聚类或主题建模这样的无监督型算法。除此之外，另一种做法是采用一些开放使用的不同数据集，然后用它来构建模型并应用。例如在同一个用例中，我们也可以直接以Web的方式来抓取一些可用的客户投诉，并将其用作我们模型中的训练数据。这也是一种很好的获取相关标签数据的替代做法。

### 有影响力的朋友

找出您社交关系图中最有影响力的人也是社交媒体中一个常见的用例。对于我们来说这个问题就是要找出一个在图结构中拥有大量入向链接和外向链接的节点，它就被认为是有影响力的节点。

在商业背景下，相同的问题可能就是要找出最有影响力的客户，并以他们为目标来推销我们的产品。

下面我们就来看一下寻找有影响力朋友的代码：

>>> friends = fb.get\_connections("me", "friends")["data"]

>>> print friends >>>for frd in friends:

>>> print fb.get\_connections(frd["id"],"friends")

一旦我们手里有了一个包含自身所有朋友、以及与这些朋友的共同朋友的列表，就可以创建一个这样的数据结构：

|  |  |  |
| --- | --- | --- |
| **源节点** | **目标节点** | **存在链接数** |
| 朋友1 | 朋友2 | 1 |
| 朋友1 | 朋友3 | 1 |
| 朋友2 | 朋友3 | 0 |
| 朋友1 | 朋友4 | 1 |

这是一种可用来生成某种网络关系的数据结构，它非常有利于对社交关系图进行可视化。我们这里使用的D3库，但Python社区中还有一个名为NetworkX（https://networkx.github.io/）的库，也可以用来生成可视化的图结构，如下图所示。要想生成可视化的图结构，我们首先要基于之前的信息弄清楚谁是谁的朋友，以此来创建一个邻接矩阵。

（图）

用D3库绘制的某样本网络的可视化图

## 本章小结

在本章，我们带您接触到了一些当下最流行的社交网络，并学习了如何使用Python来获取它们的数据。在这个过程中，我们理解了它们的数据结构和各类属性数据，并探索了其API所提供的不同选项。

另外，我们也探讨了一些在社交媒体挖掘领域中最常见的用例。这些用例包括热门话题提取、影响者检测，信息流分析等。我们还对其中的一些用例进行了可视化处理。除此之外，我们还运用了上一章所学到的知识，用NLTK来完成一些主题获取和实体提取的任务，并用scikit-learn对一些投诉信息进行了分类。

最后，我们建议读者可以再寻找一些其它的社交网络环境来执行相同的用例，并自行探索它们。这些社交网络大部分都会提供一组数据API，而且其中绝大多数是开放的，足以让我们做一些有趣的分析。如果您想实现本章所学习到的这些应用，就需要了解如何用这些API来获取数据，然后如何运用之前章节中所学到的一些概念。我们期待读者在学习这一切的过程中能想出更多的用例，和一些有趣的社交媒体分析。

1. 译者注：经译者查证，该链接已经失效，读者可以通过访问<https://dev.twitter.com/resources/twitter-libraries>来获取相关内容。 [↑](#footnote-ref-1)
2. 译者注：考虑到原文讨论的是英文环境下的文本处理，其处理逻辑跟中文大不相同，所以数据文本还是不翻译为好。 [↑](#footnote-ref-2)