# 第9章 **Python中的社交媒体挖掘**

This chapter is all about social media. Though it's not directly related to NLTK / NLP, social data is also a very rich source of unstructured text data. So, as NLP enthusiasts, we should have the skill to play with social data. We will try to explore how to gather relevant data from some of the most popular social media platforms. We will cover how to gather data from Twitter, Facebook, and so on using Python APIs. We will explore some of the most common use cases in the context of social media mining, such as trending topics, sentiment analysis, and so on.

这一章是关于社交媒体。虽然它不直接与NLTK / NLP相关，但社交数据也是非常丰富的非结构化文本数据源。所以，作为NLP爱好者，我们应该有技能来玩社交数据。我们将尝试探索如何从一些最受欢迎的社交媒体平台收集相关数据。我们将介绍如何使用Python API从Twitter，Facebook等收集数据。我们将在社交媒体挖掘的上下文中探索一些最常见的用例，例如趋势主题，情绪分析等。

You already learned a lot of topics under the concepts of natural language processing and machine learning in the last few chapters. We will try to build some of the applications around social data in this chapter. We will also provide you with some of the best practices to deal with social data, and look at social data from the context of graph visualization.

在过去几章中，您已经学习了很多关于自然语言处理和机器学习概念的主题。我们将在本章中尝试构建一些关于社交数据的应用程序。我们还将向您提供一些最佳实践来处理社交数据，并从图形可视化的上下文中查看社交数据。

There is a graph that underlies social media and most of the graph-based problems can be formulated as information flow problems and finding out the busiest node in the graph. Some of the problems such as trending topics, influencer detection, and sentiment analysis are examples of these. Let's take some of these use cases, and build some cool applications around these social networks.

有一个图形是社交媒体的基础，大多数基于图形的问题可以被制定为信息流问题和找出图中最繁忙的节点。一些问题，如趋势主题，影响者检测和情绪分析是这些的例子。让我们采取一些这些用例，并在这些社交网络周围构建一些很酷的应用程序。

By the end of this chapter,:

到本章结束时：

* You should be able to collect data from any social media using APIs.
* You will also learn to formulate the data in a structured format and how to build some amazing applications.
* Lastly, we will be able to visualize and gain meaningful insight out of social media.
* 您应该能够使用API从任何社交媒体收集数据。
* 您还将学习以结构化格式来制定数据，以及如何构建一些了不起的应用程序。
* 最后，我们将能够从社交媒体中可视化并获得有意义的洞察力。

## 数据收集

The most important objective of this chapter is to gather data across some of the most common social networks. We will look mainly at Twitter and Facebook and try to give you enough details about the API and how to effectively use them to get relevant data. We will also talk about the data dictionary for scrapped data, and how we can build some cool apps using some of the stuff we learned so far.

本章最重要的目标是收集跨越一些最常见的社交网络的数据。 我们将主要看看Twitter和Facebook，并尝试向您提供有关API的足够详细信息，以及如何有效地使用它们来获取相关数据。 我们还将讨论关于已废弃数据的数据字典，以及我们如何使用我们迄今为止学到的一些东西来构建一些很酷的应用程序。

### Twitter

We will start with one of the most popular and open social media that is completely public. This means that practically, you can gather entire Twitter stream, which is payable, while you can capture one percent of the stream for free. In the context of business, Twitter is a very rich resource of information such as public sentiments and emerging topics.

我们将从一个完全公开的最流行和开放的社交媒体之一开始。 这意味着，实际上，你可以收集整个Twitter流，这是应付，而你可以捕获的流的免费的百分之一。 在商业背景下，Twitter是一个非常丰富的信息资源，如公众情绪和新兴话题。

Let's get directly to face the main challenge of getting the tweets relevant to your use case.

让我们直接面对的主要挑战，让tweets与您的用例相关。

|  |
| --- |
| The following is the repository of many Twitter libraries. These libraries are not verified by Twitter, but run on the Twitter API.  <https://dev.twitter.com/overview/api/twitterlibraries> |

There are more than 10 Python libraries there. So pick and choose the one you like. I generally use Tweepy and we will use it to run the examples in this book. Most of the libraries are wrappers around the Twitter API, and the parameters and signatures of all these are roughly the same.

有超过10个Python库。 所以选择你喜欢的一个。 我一般使用Tweepy，我们将使用它来运行本书中的示例。 大多数库是围绕Twitter API的包装器，所有这些的参数和签名大致相同。

The simplest way to install Tweepy is to install it using pip:

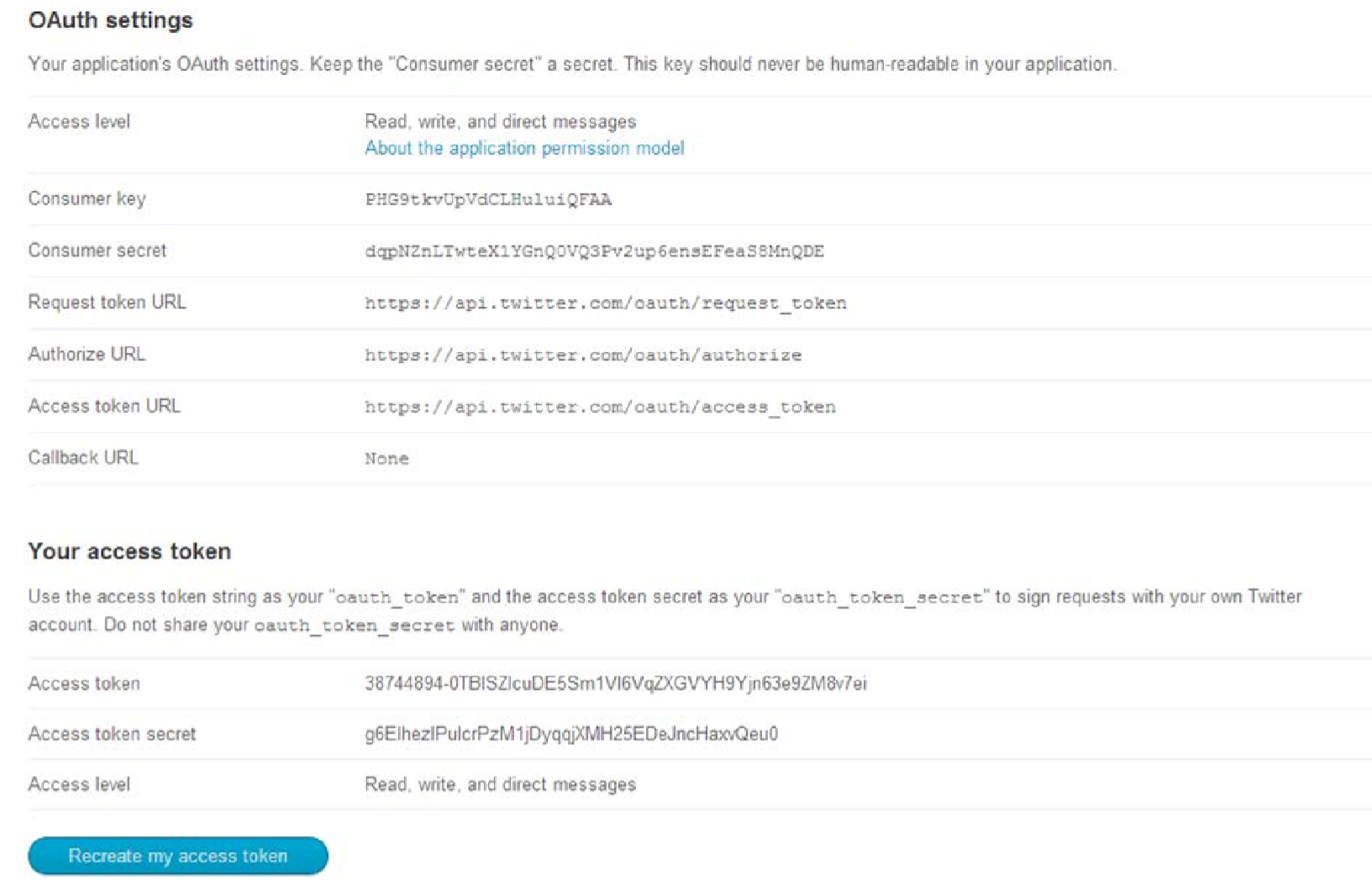
安装Tweepy的最简单的方法是使用pip安装：

$ pip install tweepy

|  |
| --- |
| The hard way is to build it from source. The GitHub link to Tweepy is:  https://github.com/tweepy/tweepy. |

To get Tweepy to work, you have to create a developer account with Twitter and get the access tokens for your application. Once you complete this, you will get your credentials and below these, the keys. Go through [https://apps.twitter.com/ app/new](https://apps.twitter.com/app/new) for registration and access tokens. The following snapshot shows the access tokens:

要使Tweepy正常工作，您必须使用Twitter创建一个开发人员帐户，并为您的应用程序获取访问令牌。 一旦你完成这个，你会得到你的凭据和以下这些，钥匙。 通过https://apps.twitter.com/ app / new注册和访问令牌。 以下快照显示访问令牌：



We will start with a very simple example to gather data using Twitter's streaming API. We are using Tweepy to capture the Twitter stream to gather all the tweets related to the given keywords:

我们将从一个非常简单的示例开始，使用Twitter的流API收集数据。 我们使用Tweepy捕获Twitter流以收集与给定关键字相关的所有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'])

In the preceding code, we used the same code given in the example of Tweepy, with a little modification. This is an example where we use the streaming API of Twitter, where we track **Apple Watch**. Twitter's streaming API provides you the facility of conducting a search on the actual Twitter stream and you can consume a maximum of one percent of the stream using this API.

在上面的代码中，我们使用了Tweepy示例中给出的相同的代码，稍作修改。这是一个例子，我们使用Twitter的流API，我们跟踪Apple Watch。 Twitter的流API提供您在实际的Twitter流上进行搜索的功能，您可以使用此API消耗流的最大百分之一。

In the preceding code, the main parts that you need to understand are the first and last four lines. In the initial lines, we are specifying the access tokens and other keys that we generated in the previous section. In the last four lines, we create a listener to consume the stream. In the last line, we use stream.filter to filter Twitter for keywords that we have put in the track. We can specify multiple keywords here. This will result in all the tweets that contain the term Apple Watch for our running example.

在上面的代码中，你需要理解的主要部分是第一行和最后四行。在初始行中，我们指定了我们在上一节中生成的访问令牌和其他密钥。在最后四行中，我们创建一个监听器来使用流。在最后一行，我们使用stream.filter来过滤我们已经放在轨道中的关键字。我们可以在这里指定多个关键字。这将导致所有的tweets包含术语Apple Watch为我们的运行示例。

In the following example, we will load the tweets we have collected, and have a look at the tweet structure and how to extract meaningful information from it. A typical tweet JSON structure looks similar to:

在下面的示例中，我们将加载我们收集的tweets，并查看tweet结构以及如何从中提取有意义的信息。典型的tweet JSON结构类似于：

{

"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

Some of the most commonly used fields of interest in data extraction are:

数据提取中最常用的一些领域是：

* text: This is the content of the tweet provided by the user
* user: These are some of the main attributes about the user, such as username, location, and photos
* Place: This is where the tweets are posted, and also the geo coordinates
* Entities: Effectively, these are the hashtags and topics that a user attaches to his / her tweets
* text：这是用户提供的tweet的内容
* user：这些是用户的一些主要属性，如用户名，位置和照片
* 地点：这是发布tweet的地方，也是地理坐标
* 实体：有效地，这些是用户附加到他/她的tweets的主题标签和主题

Every attribute in the previous figure can be a good use case for some of the social mining exercises done in practice. Let's jump onto the topic of how we can get to these attributes and convert them to a more readable form, or how we can process some of these:

上图中的每个属性都可以成为实践中进行的一些社会挖掘练习的一个很好的用例。 让我们跳转到我们如何获得这些属性并将它们转换为更易读的形式，或者我们如何处理这些属性的主题：

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

The output of the preceding code will give you, as expected, four lists in which all the tweet content is in tweet\_texts and the location of the tweets and hashtags.

前面的代码的输出将如所期望地给出四个列表，其中所有的tweet内容在tweet\_texts中以及tweet和hashtag的位置。

|  |
| --- |
| In the code, we are just loading a JSON output generated using json. loads(). I would recommend you to use an online tool such as Json Parser (http://json.parser.online.fr/) to get an idea of what your JSON looks like and what are its attributes (key and value).  在代码中，我们只是加载使用json生成的JSON输出。 loads（）。 我建议你使用一个在线工具，如Json Parser（http://json.parser.online.fr/）来了解你的JSON是什么样子，它的属性（键和值）是什么。 |

Next, if you look, there are different levels in the JSON, where some of the attributes such as text have a direct value, while some of them have more nested information. This is the reason you see, where when we are looking at hashtags, we have to iterate one more level, while in case of text, we just fetch the values. Since our file actually has a list of tweets, we have to iterate that list to get all the tweets, while each tweet object will look like the example tweet structure.

接下来，如果你看，JSON中有不同的级别，其中一些属性，如文本有一个直接的值，而其中一些有更多的嵌套信息。 这是你看到的原因，当我们查看hashtag时，我们必须重复一个级别，而在文本的情况下，我们只是获取值。 由于我们的文件实际上有一个tweets列表，我们必须迭代该列表以获取所有的tweets，而每个tweet对象将看起来像示例tweet结构。

### 热门话题

Now, if we look for trending topics in this kind of a setup. One of the simplest ways to find them could be to look for frequency distribution of words across tweets. We already have a list of tweet\_text that contains the tweets:

现在，如果我们在这种设置中查找趋势主题。 找到它们的最简单的方法之一可以是查找单词在tweet中的频率分布。 我们已经有一个包含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)

One other more complex way of doing this could be the use of the part of speech tagger that you learned in *Chapter 3*, *Part of Speech Tagging*. The theory is that most of the time, topics will be nouns or entities. So, the same exercise can be done like this. In the preceding code, we read every tweet and tokenize it, and then use POS as a filter to only select nouns as topics:

另一个更复杂的方法是使用您在第3章“语音标记”中学到的语音标记器部分。 理论是，大多数时候，主题将是名词或实体。 所以，同样的做法可以这样做。 在上面的代码中，我们读取每个tweet并对其进行tokenize，然后使用POS作为过滤器，仅选择名词作为主题：

>>>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

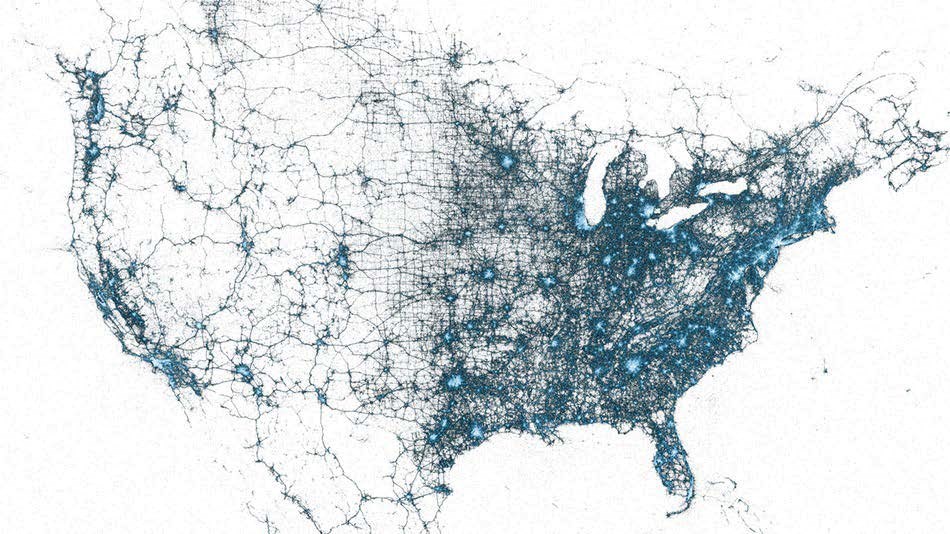
If we want to see a much cooler example, we can gather tweets across time and then generate plots. This will give us a very clear idea of the trending topics. For example, the data we are looking for is "Apple Watch". This word should peak on the day when Apple launched Apple Watch and the day they started selling it. However, it will be interesting to see what kind of topics emerged apart from those, and how they trended over time.

如果我们想看到一个更酷的例子，我们可以收集tweets跨时间，然后生成图。 这将给我们一个非常清楚的想法的热门话题。 例如，我们正在寻找的数据是“Apple Watch”。 这个词应该在苹果推出Apple Watch的那一天和他们开始销售它的那一天达到高峰。 然而，看到除了这些主题之外出现了什么样的主题，以及它们随着时间的推移如何趋势将是有趣的。

## 地理可视化

One of the other common application of social media is geo-based visualization. In the tweet structure, we saw attributes named geo, longitude, and latitude. Once you have access to these values, it is very easy to use some of the common visualization libraries, such as **D3**, to come up with something like this:

社交媒体的另一个常见应用之一是基于地理的可视化。 在tweet结构中，我们看到了名为geo，longitude和latitude的属性。 一旦你可以访问这些值，很容易使用一些常见的可视化库，如D3，想出这样的东西：



This is just an example of what we can achieve with these kind of visualizations; this was the visualization of a tweet in the U.S. We can clearly see the areas of increased intensity in eastern places such as New York. Now, a similar analysis done by a company on the customers can give a clear insight about which are some of the most popular places liked by our customer base. We can text mine these tweets for sentiment, and then we can infer insights about customers as to in which states they are not happy with the company and so on.

这只是我们可以用这些可视化实现的一个例子; 这是在美国的推文的可视化。我们可以清楚地看到东部地区，如纽约的强度增加的地区。 现在，一家公司对客户进行的类似分析可以清楚地了解我们客户群喜欢的一些最受欢迎的地方。 我们可以根据情绪对这些推文进行文本挖掘，然后我们可以推断有关客户的洞察以及他们对公司不满意的状态等等。

### Influencers detection

Detection of important nodes in the social graph that has a lot of importance is another great problem in the context of social graphs. So, if we have millions of tweets streaming about our company, then one important use case would be to gather the most influential customers in the social media, and then target them for branding, marketing, or improving customer engagement.

在社交图中检测具有很大重要性的重要节点是社交图的上下文中的另一个大问题。 因此，如果我们有数百万条关于我们公司的推文，那么一个重要的用例就是收集社交媒体中最有影响力的客户，然后针对他们进行品牌推广，营销或改善客户参与度。

In the case of Twitter, this goes back to the graph theory and concept of PageRank, where in a given node, if the ratio of outdegree and indegree is high, then that node is an influencer. This is very intuitive since people who have more followers than the number of people they follow are typically, influencers. One company, **KLOUT,** [(https://klout.com/)](https://klout.com/) has been focusing on a similar problem. Let's write a very basic and intuitive algorithm to calculate Klout's score:

在Twitter的情况下，这回到PageRank的图论和概念，其中在给定节点中，如果outdegree和indegree的比率高，则该节点是影响者。 这是非常直观的，因为拥有更多关注者的人比他们关注的人的数量通常，影响者。 一家公司，KLOUT，（https://klout.com/）一直专注于类似的问题。 让我们写一个非常基本和直观的算法来计算Klout的分数：

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

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

Some of the examples where we worked on Twitter will hold exactly the same modification of content field. We can build a trending topic example with Facebook posts. We can also visualize Facebook users and post on the **geomap** and influencer kind of use cases. In fact, in the next section, we will see a variation of this in the context of Facebook.

我们在Twitter上工作的一些示例将保持完全相同的内容字段修改。 我们可以用Facebook帖子构建一个热门话题示例。 我们还可以可视化Facebook用户和发布在地理地图和影响者类的用例。 事实上，在下一节中，我们将在Facebook的上下文中看到这种变化。

### Facebook

Facebook is a bit more personal, and somewhat private social network. Facebook does not allow you to gather the feeds/posts of the user simply for security and privacy reasons. So, Facebook's graph API has a limited way of accessing the feeds of the given page. I would recommend you to go to [https://developers.facebook. com/docs/graph-api/using-graph-api/v2.3](https://developers.facebook.com/docs/graph-api/using-graph-api/v2.3) for better understanding.

Facebook有点更个人化，有点私人社交网络。 Facebook不允许您收集用户的饲料/帖子只是为了安全和隐私的原因。 因此，Facebook的图形API具有访问给定页面的馈送的有限方式。 我建议你去https：//developers.facebook。 com / docs / graph-api / using-graph-api / v2.3以便更好地理解。

The next question is how to access the Graph API using Python and how to get started with it. There are many wrappers written over Facebook's API, and we will use one the most common Facebook SDK:

下一个问题是如何使用Python访问Graph API以及如何开始使用它。 有很多封装在Facebook的API上编写，我们将使用最常见的Facebook SDK：

$ pip install facebook-sdk

|  |
| --- |
| You can also install it through: https://github.com/Pythonforfacebook/facebook-sdk. |

The next step is to get the access token for the application while Facebook treats every API call as an application. Even for this data collection step, we will pretend to be an application.

下一步是获取应用程序的访问令牌，而Facebook将每个API调用视为一个应用程序。 即使对于这个数据收集步骤，我们也会假装成一个应用程序。

|  |
| --- |
| To get your token, go to: https://developers.facebook.com/tools/explorer. |

We are all set now! Let's start with one of the most widely used Facebook graph APIs. In this API, Facebook provides a graph-based search for pages, users, events, places, and so on. So, the process of getting to the post becomes a two-stage process, where we have to look for a specific pageid / userid related to our topic of interest, and then we will be able to access the feeds of that page. One simple use case for this kind of an exercise could be to use the official page of a company and look for customer complaints. The way to go about this is:

我们都设置了！ 让我们从一个最广泛使用的Facebook图形API开始。 在此API中，Facebook提供了一个基于图表的搜索页面，用户，事件，地点等。 因此，到达帖子的过程变成了两个阶段的过程，我们必须寻找与我们感兴趣的主题相关的特定的pageid / userid，然后我们将能够访问该页面的提要。 这种练习的一个简单的用例可以是使用公司的官方页面并寻找客户投诉。 要做到这一点的方法是：

>>>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))

The code will attach the token to the Facebook Graph API and then we will make a REST call to Facebook. The problem with this is that we have to have the ID of the given page with us beforehand. The code which will attach the token is as follows:

代码将把令牌附加到Facebook Graph API，然后我们将对REST进行REST调用。 这个问题是我们必须事先获得给定页面的ID。 将附加令牌的代码如下：

"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

},

Here, we showed a similar schema for the Facebook data as we did for Twitter, and now we can see what kind of information is required for our use case. In most of the cases, the user post, category, name, about, and likes are some of the important fields. In this example, we are showing a page of a museum, but in a more businessdriven use case, a company page has a long list of posts and other useful information that can give some great insights about it.

在这里，我们为Twitter数据显示了类似的Facebook数据模式，现在我们可以看到我们的用例需要什么样的信息。在大多数情况下，用户post，category，name，about和likes是一些重要的字段。在这个例子中，我们展示了一个博物馆的页面，但在一个更商业化的用例中，公司页面有一个长列表的帖子和其他有用的信息，可以提供一些伟大的见解。

Let's say I have a Facebook page for my organization xyz.org and I want to know about the users who complained about me on the page; this is good for a use case such as complaint classification. The way to achieve the application now is simple enough. You need to look for a set of keywords in fdump.txt, and it can be as complex as scoring using a text classification algorithm we learned in *Chapter 6, Text Classification*.

假设我有一个Facebook页面给我的组织xyz.org，我想知道在页面上投诉我的用户;这对于诸如投诉分类的用例是好的。现在实现应用程序的方法很简单。您需要在fdump.txt中查找一组关键字，它可以像使用我们在第6章“文本分类”中学到的文本分类算法那样复杂。

The other use case could be to look for a topic of interest, and then to look for the resulting pages for open posts and comments. This is exactly analogous to searching using the graph search bar on your Facebook home page. However, the power of doing this programmatically is that we can conduct these searches and then each page can be recursively parsed for use comments. The code for searching user data is as follows: User search

另一个用例可以是查找感兴趣的主题，然后查找结果页面以查找打开的帖子和注释。这完全类似于使用Facebook主页上的图形搜索栏进行搜索。但是，以编程方式这样做的好处是我们可以进行这些搜索，然后每个页面可以递归解析使用注释。搜索用户数据的代码如下：用户搜索

>>>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'})

Once we have dumped all the relevant data into a structured format, we can apply some of the concepts we learned when we went through the topics of NLP and machine learning. Let's pick the same use case of finding posts, that will mostly be complaints on a Facebook page.

一旦我们将所有相关数据转换为结构化格式，我们可以应用我们在学习NLP和机器学习主题时学到的一些概念。 让我们选择相同的用例来发现帖子，这将主要是在Facebook页面上的投诉。

I assume that we now have the data in the following format:

我假设我们现在有以下格式的数据：

|  |  |
| --- | --- |
| **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 |

We will go back to the same example we had in *Chapter 6*, *Text Classification*, where we built a text classifier to detect whether the **SMS** (text message) was spam. Similarly, we can create training data using this kind of data, where from the given set of posts, we will ask manual taggers to tag the comments that are complaints and the ones that are not. Once we have significant training data, we can build the same text classifier:

我们将回到我们在第6章“文本分类”中所使用的相同示例，其中我们构建了一个文本分类器来检测SMS（文本消息）是否为垃圾邮件。 类似地，我们可以使用这种数据创建训练数据，其中从给定的帖子集合，我们将要求手动标记器标记作为投诉的评论，而不是标记。 一旦我们有重要的训练数据，我们可以构建相同的文本分类器：

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)

Let's assume that these three are the only samples. We can tag 1st and 3rd to be classified as complaints, while 2nd will not be a complaint. Although we will build a vectorizer of unigram and bigram in the same way, we can actually build a classifier using the same process. I ignored some of the preprocessing steps here. You can use the same process as discussed in *Chapter 6*, *Text Classification*. In some of the cases, it will be hard/expensive to get training data like this. In some of these cases, we can apply either an unsupervised algorithm, such as text clustering or topic modeling. The other way is to use some different dataset that is openly available and build model on that and apply it here. For example, in the same use case, we can crawl some of the customer complaints available on the Web and use that as training data for our model. This can work as a good proxy for labeled data.

让我们假设这三个是唯一的样本。 我们可以标记第一和第三被归类为投诉，而第二不被投诉。 虽然我们将以相同的方式构建一个unigram和bigram的向量化器，我们实际上可以使用相同的过程构建一个分类器。 我忽略了这里的一些预处理步骤。 您可以使用与第6章“文本分类”中讨论的相同的过程。 在某些情况下，得到这样的训练数据将是困难/昂贵的。 在其中一些情况下，我们可以应用无监督算法，例如文本聚类或主题建模。 另一种方法是使用一些公开可用的不同数据集，并构建模型并在此处应用它。 例如，在相同的用例中，我们可以抓取Web上可用的一些客户投诉，并将其用作我们模型的培训数据。 这可以作为标记数据的良好代理。

### Influencer friends

One other use case of social media could be finding out the most influencer in your social graph. In our case, it could be finding out a clear node that has a vast amount of inlinks and outlinks will be the influencer in the graph.

社交媒体的另一个用例可能是发现你的社交图中最有影响力的人。 在我们的例子中，它可能是发现一个明确的节点，有大量的链接和外链将成为图中的影响者。

The same problem in the context of business can be finding out the most influential customers, and targeting them to market our products.

在商业背景下同样的问题可以找出最有影响力的客户，并瞄准他们推销我们的产品。

The code for the Influencer friends is as follows:

Influencer朋友的代码如下：

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

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

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

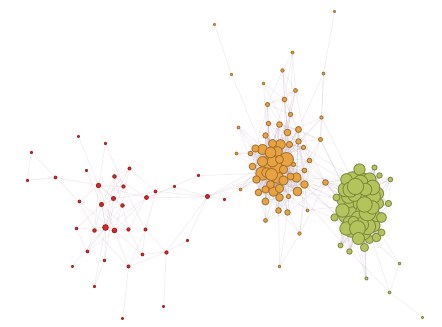
Once you have a list of all your friends and mutual friends, you can create a data structure like this:

一旦你有一个所有的朋友和共同的朋友的列表，你可以创建一个数据结构，像这样：

|  |  |  |
| --- | --- | --- |
| **source node** | **destination node** | **link\_exist** |
| Friend 1 | Friend 2 | 1 |
| Friend 1 | Friend 3 | 1 |
| Friend 2 | Friend 3 | 0 |
| Friend 1 | Friend 4 | 1 |

This a kind of data structure that can be used to generate a network, and is a very good way of visualizing the social graph. I used D3 here, but python also has a library called **NetworkX** (<https://networkx.github.io/>) that can be used to generate graph visualization, as shown in the following graph. To generate a visualization, you need to arrive at a adjacency matrix that can be created based on the bases of the preceding information about who is the friend of whom.

这是一种可用于生成网络的数据结构，是一种可视化社交图的非常好的方法。 我在这里使用D3，但是python还有一个名为NetworkX（https://networkx.github.io/）的库，可以用来生成图形可视化，如下图所示。 要生成可视化，您需要到达一个邻接矩阵，可以基于前面的信息的基础来创建谁是谁的朋友。



Visualization of a sample network in D3

## 本章小结

In this chapter, we touched upon some of the most popular social networks. You learned how to get data using Python. You understood the structure and kind of attributes data has. We explored different options provided by the API.

在本章中，我们谈到了一些最流行的社交网络。您学习了如何使用Python获取数据。你理解数据的结构和种类。我们探索了API提供的不同选项。

We explored some of the most common use cases in the context of social media mining. We touched upon the use cases about trending topics, influencer detection, information flow, and so on. We visualized some of these use cases. We also applied some of the learnings from the previous chapter, where we used NLTK to get some of the topic and entity extraction, while in scikit-learn we classified some of the complaints.

我们探讨了一些在社交媒体挖掘的上下文中最常见的用例。我们讨论了关于热门话题，影响者检测，信息流等的使用案例。我们可视化了一些这些用例。我们还应用了前一章的一些学习，我们使用NLTK来获得一些主题和实体提取，而在scikit-learn中，我们分类了一些投诉。

In conclusion, I would suggest that you look for some of the same use cases in context of some other social networks and try to explore them. The great part of these social networks is that all of them have a data API, and most of them are open enough to do some interesting analysis. If you apply the same learning you did in this chapter, you need to understand the API, how to get the data, and then how to apply some of the concepts we learned in the previous chapters. I hope that after learning all this, you will come up with more use cases, and some interesting analysis of social media.

总之，我建议你在一些其他社交网络的上下文中寻找一些相同的用例，并尝试探索它们。这些社交网络的很大一部分是，他们都有一个数据API，其中大多数是开放的，足以做一些有趣的分析。如果应用与本章中相同的学习，您需要了解API，如何获取数据，然后如何应用我们在前面章节中学到的一些概念。我希望在学习所有这些，你会想出更多的用例，和一些有趣的社交媒体分析。