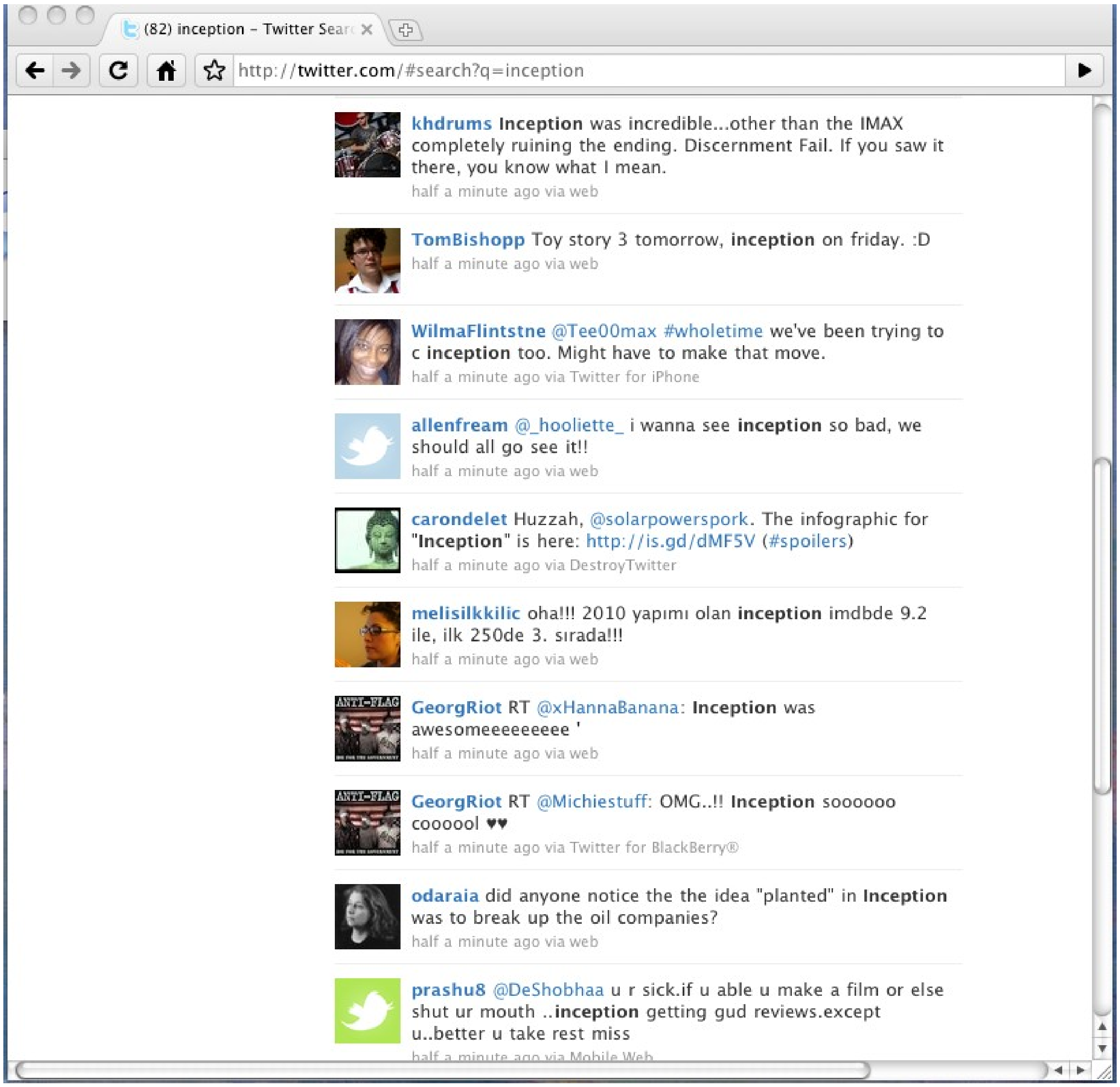
Chapter 6: Naïve Bayes and unstructured text

In previous chapaters we've looked at recommendation systems that have people explicitly rate things with star systems (5 stars for Phoenix), thumbs-up/thumbs-down (Inception-- thumbs-up!), and numerical scales. We've looked at implicit things like the behavior of people—did they buy the item, did they click on a link. In this chapter we look at the old skool idea of deciding on whether a person likes or dislikes something based on what they say or write. For example, consider the following Twitter messages:


(82) inception – Twitter Searc
http://twitter.com/#search?q=inception

khdrums **Inception** was incredible...other than the IMAX completely ruining the ending. Discernment Fail. If you saw it there, you know what I mean.
half a minute ago via web

TomBishopp Toy story 3 tomorrow, **inception** on friday. :D
half a minute ago via web

WilmaFlintstne @Tee00max #wholetime we've been trying to c **inception** too. Might have to make that move.
half a minute ago via Twitter for iPhone

allenfream @_hooliette_ i wanna see **inception** so bad, we should all go see it!!
half a minute ago via web

carondelet Huzzah, @solarpowerspork. The infographic for "**Inception**" is here: http://is.gd/dMF5V (#spoilers)
half a minute ago via DestroyTwitter

melisilkkilic oha!!! 2010 yapımı olan **inception** imdbde 9.2 ile, ilk 250de 3. sırada!!!
half a minute ago via web

GeorgRiot RT @xHannaBanana: **Inception** was awesomeeeeeeeee '
half a minute ago via web

GeorgRiot RT @Michiestuff: OMG..!! **Inception** soooooo coooool ♥♥
half a minute ago via Twitter for BlackBerry®

odaraia did anyone notice the the idea "planted" in **Inception** was to break up the oil companies?
half a minute ago via web

prashu8 @DeShobhaa u r sick.if u able u make a film or else shut ur mouth ..**inception** getting gud reviews.except u..better u take rest miss
half a minute ago via Mobile Web

We, as speakers of English can see that khdrums liked Inception since he said it was '*incredible*'; we gather that GeorgRiot liked it because he says '*awesomeeeeeee*'; and '*soooooo cooool ♥♥*'. We also know that even though the user allenfream used the word 'bad' they are not saying tjey did not like the movie.

Suppose I am at my local food co-op and see something called Chobani Greek Yogurt. It looks interesting but is it any good? I get out my iPhone, do a google search and find the following from the blog "*Woman Does Not Live on Bread Alone*":

> Chobani nonfat greek yogurt.
>
> Have you ever had greek yogurt? If not, stop reading, gather your keys (and a coat if you live in New York) and get to your local grocery. Even when nonfat and plain, greek yogurt is so thick and creamy, I feel guilty whenever I eat it. It is definitely what yogurt is MEANT to be. The plain flavor is tart and fantastic. Those who can have it, try the honey version. There's no sugar, but a bit of honey for a taste of sweetness (or add your own local honey-- local honey is good for allergies!). I must admit, even though I'm not technically supposed to have honey, if I've had a bad day, and just desperately need sweetness, I add a teaspoon of honey to my yogurt, and it's SO worth it. The fruit flavors from Chobani all have sugar in them, but fruit is simply unnecessary with this delicious yogurt. If your grocery doesn't carry the Chobani brand, Fage (pronounced Fa-yeh) is a well known, and equally delicious brand.
>
> Now, for Greek yogurt, you will pay about 50 cents to a dollar more, and there are about 20 more calories in each serving. But it's worth it, to me, to not feel deprived and saddened over an afternoon snack!

http://womandoesnotliveonbreadalone.blogspot.com/2009/03/sugar-free-yogurt-reviews.html

Is that a positive or negative review for Chobani or a negative one? Even based on the second sentence: If not, stop reading, gather your keys … and get to your local grocery store, it seems that this will be a positive review. She describes the flavor as *fantastic* and calls the yogurt *delicious*. It seems that I should buy it and check it out.

An automatic system for determining positive and negative texts.

Let's imagine an automatic system that can read some text and decide whether it is a positive or negative report about a product. Why would we want such a system? Suppose there is a company that sells health monitors, they might want to know about what people are saying about their products. Are what people say mostly positive or negative? They release an ad campaign for a new product. Are people favorable about the product (*Man, I sooo want this!*) or negative (*looks like crap*). Apple has a press conference to talk about the iPhone problems. Is the resulting press coverage positive? A Senate candidate delivers a major policy speech—do the political bloggers view it favorably? So an automatic system does sound useful.

So how can we create it? Let's say I want to create a system that can tell whether a person likes or dislikes various food products. We might come up with an idea of having a list of words that would provide evidence that a person likes the product and another list of words that provides evidence that the person doesn't like the product.

Like	**Dislike**
delicious	awful
tasty	bland
good	bad
love	hate
smooth	gritty
awesome	sucks

We can just count the number of like words in the text and the dislikes and classify it based on which count is higher. We can do this for other classification tasks. For example, if we want to decide whether someone is pro-choice or pro-life, we can base it on the words and phrases they use. If they use the phrase '*unborn child*' then chances are they are pro-life; if they use *fetus* they are more likely to be pro-choice. It's not surprising that we can use the occurrence of words to classify text.

Rather than just using raw counts to classify text, we can use the naïve Bayes methods that were introduced in the previous chapter.

naïve Bayes training

In the naïve Bayes method we will have a training corpus—a collection of documents. Consider the task of identifying whether the writer of a document likes or dislikes a food product. The documents in the training corpus will be tagged as 'like' documents (documents that reflect a positive review) and 'dislike documents'. The naïve Bayes formula for classifying a document is

$$h_{MAP} = argmax_{h \in H} P(D|h)P(h)$$

Here the hypotheses, *h*, are like and dislike. For the *P(h)* component of the formula we need to compute *P(like)* and *P(dislike)*. *P(like)* is simply the number of documents tagged as '*like*' divided by the total number of documents. If I have 100 documents in my training set, 60 are tagged as *like* and 40 as *dislike* then *P(like)* = .6 and *P(dislike)* = .4.

Now let's examine the $P(D|h)$ part of the formula—the probability of seeing some evidence, some data *D* given the hypothesis *h*. The data *D* we are going to use is the words in the text. One approach would be to start with the first sentence of a document, for example, *Have you ever had greek yogurt?* And compute things like the probability that a 'like' document starts with the word *Have*; what's the probability of a 'like' document having a second word of *you*; and so on. Let's say each document is only 100 words long, and our vocabulary is 50,000 words. That means that we would be computing 10 million probabilities for the like and dislike hypotheses. That's a lot of probabilities and it makes this approach unworkable. We are going to simplify things a bit by treating the documents as bags of unordered words. Instead of asking things like *What's the probability that the third word is* love *given it is a 'like' document* we will ask *What's the probability that the word* love *occurs in a 'like' document.*

We are going to compute this as follows. For each hypothesis (in this case 'like' and dislike')

1. combine the documents tagged with that hypothesis into one text file.
2. Determine the vocabulary (the list of unique words in the text file).
3. For each word in the vocabulary w_k count how many times it occurred in the text - n_k.
4. Compute the probability of the word in the vocabulary w_k as follows

$$P(w_k|h_i) = \frac{n_k + 1}{n + |Vocabulary|}$$

where n is the number of word instances in the text and $|Vocabulary|$ is the number of unique words in the text.

naïve Bayes classification

Once we built the classifier we can classify a document using the formula

$$prediction = argmax_{v_j \in V} P(v_j) \prod_{i \in positions} P(a_i | v_j)$$

Newsgroup corpus

We will first investigate how this algorithm works by using a standard reference corpus of usenet newsgroup posts. The data consists of posts to 20 newsgroups:

comp.graphics	misc.forsale	soc.religion.christian	alt.atheism
comp.os.ms-windows.misc	rec.autos	talk.politics.guns	sci.space
comp.sys.ibm.pc.hardware	rec.motorcycles	talk.politics.mideast	sci.crypt
comp.sys.mac.hardware	rec.sport.baseball	talk.politics.misc	sci.electronics
comp.windows.x	rec.sport.hockey	talk.religion.misc	sci.med

We would like to build a classifier than can correctly determine what group the post came from. The data is available on the website for the book, http://guidetodatamining.com.

Throwing things out

Consider this newsgroup task in more detail. Just based on the words in a post, we are going to attempt to tell which newsgroup the post is from. For example to determine the following post is from soc.religion.christian:

> I'm new to this group, and maybe this has been covered already, but does anybody out there see the current emphasis on the environment being turned (unintentionally, of course) into pantheism?
>
> I've debated this quite a bit, and while I think a legitimate concern for the planet is a great thing, I can easily see it being perverted into something dangerous.
>
> As evidence, may I quote THE WALL STREET JOURNAL (of all things!), April 2 (Editorial page):
>
> "We suspect that's because one party to the (environmental) dispute thinks the Earth is

> sanctified. It's clear that much of the environmentalist energy is derived from what has been called the Religious Left, a SECULAR, or even PAGAN fanaticism that now WORSHIPS such GODS as nature and gender with a reverence formerly accorded real religions." (EMPHASIS MINE).

I want to draw your attention to the words in the above text that are common words in English—the ones that occur most frequently. Let's say I remove say any occurrences of the 200 most frequent words in English before I analyze the text:

> I'm new to this group, and maybe this has been covered already, but does anybody out there see the current emphasis on the environment being turned (unintentionally, of course) into pantheism?
>
> I've debated this quite a bit, and while I think a legitimate concern for the planet is a great thing, I can easily see it being perverted into something dangerous.
>
> As evidence, may I quote THE WALL STREET JOURNAL (of all things!), April 2 (Editorial page):
>
> "We suspect that's because one party to the (environmental) dispute thinks the Earth is

It doesn't look like removing these words will have any impact on our ability to categorize texts. Indeed data miners have called such words *words without any content,* and *fluff words*. H.P. Luhn, in his seminal paper 'The automatic creation of literature abstracts' says of these words that they are "too common to have the type of significance being sought and would constitute 'noise' in the system." That *noise* argument is interesting as it implies that removing these words will improve performance. These words that we remove are called 'stop words'. We have a list of such words, the 'stopword list', and remove these words from the text in a preprocessing step. We remove these words because 1) it cuts down on the amount of processing we need to do and 2) it does not negatively impact the performance of our system—as the *noise* argument suggests removing them might improve performance.

The hazards of stopword removal

While removing stopwords might be a good idea in some situations, the claim that the words have no content and consitute *noise* turns out to be false. For example, it turns out just using the stopwords and throwing out the rest (the reverse technique of the above) provides sufficient information to identify where Arabic documents were written. (If you are curious about this check out a paper *Linguistic Dumpster Diving: Geographical Classification of Arabic Text* I co-wrote with some of my colleagues at New Mexico State University. It is available on my website http://zacharski.org. Here's a simple example of the importance of these common words. Suppose my Google search is "*flights to Austin from El Paso*." If I delete common words before processing I get: '*flights Austin El Paso*'--clearly I lost a significant amount of the meaning. Many current text analysis systems do not use stopwords. Google does not remove stop words when indexing the web. However, for this newsgroup classification task, I am going to go ahead and remove stop words. If you are going to do text based classification for a particular problem, do not just automatically implement stop word removal without thinking. Think through whether stop word removal is a good idea for your particular task.

Back to the newsgroups

The newsgroup corpus is available from http://people.csail.mit.edu/jrennie/20Newsgroups/. What we want to do in general is train on a portion of our corpus and then use the remainder to test our classifier. In this case, I randomly selected 700 documents from each newsgroup to be in the training set. There were 2 newsgroups that did not have 700 documents in the initial set. In those cases I randomly selected 600 documents to be in the training set. The remainder of the documents I put in the test set. This data is available on the book website.

Python code

First, let's consider the code for the training part of the naïve Bayes classifier. I am going to assume the data is organized as follows:

```
TrainingFiles
       alt.atheism
              text file 1 for alt.atheism
              text file 2
              …
              text file n
       comp.graphics
              text file 1 for comp.graphics
              ...
```

That is, I am going to have a directory (in this example, it is called TrainingFiles). Underneath that directory I have subdirectories representing the different classification categories (in this case alt.atheism, comp.graphics, etc.). The names of these subdirectories match the category names. Within those subdirectories are the text files that represent the training data for each category. The test file directory is organized in a similar way. In the code below, the function `train`, takes as arguments, the training directory (in this case ~/TrainingFiles/) and the name of a category. It reads all the files in that categories subdirectory—counting word occurrences. The init method calls train for each category. After all the counts are done, it computes the probabilities. For example it computes P(*god*| alt.atheism)--the probability that the word *god* appears in the text files appearing in the alt.atheism newsgroup. Recall from above that this formula is

$$P(w_k|h_i)=\frac{n_k+1}{n+|Vocabulary|}$$

```
import os, codecs, math

class BayesText:

   def __init__(self, trainingdir, stopwordlist):
        """This class implements a naive Bayes approach to text classification
trainingdir  is the training data. Each subdirectory of trainingdir is titled with
       the name of the classification category -- those subdirectories in turn contain the
       text files for that category.
       The stopwordlist is a list of words (one per line) will be removed before any
       counting takes place.
       """
       self.vocabulary = {}
       self.prob = {}
       self.totals = {}
       self.stopwords = {}
       f = open(stopwordlist)
       for line in f:
          self.stopwords[line.strip()] = 1
       f.close()
       categories = os.listdir(trainingdir)
       #filter out files that are not directories
       self.categories = [filename for filename in categories if os.path.isdir(trainingdir +
                         filename)]
       print("Counting ...")
       for category in self.categories:
          print('    ' + category)
          (self.prob[category], self.totals[category]) = self.train(trainingdir, category)
       # I am going to eliminate any word in the vocabulary that doesn't occur at least 3
       # times
       toDelete = []
       for word in self.vocabulary:
          if self.vocabulary[word] < 3:
             # mark word for deletion
             # can't delete now because you can't delete from a list you are currently
             # iterating over
             toDelete.append(word)
       # now delete
       for word in toDelete:
          del self.vocabulary[word]
       # now compute probabilities
       vocabLength = len(self.vocabulary)
       print("Computing probabilities:")
       for category in self.categories:
          print('    ' + category)
          denominator = self.totals[category] + vocabLength
          for word in self.vocabulary:
             if word in self.prob[category]:
                count = self.prob[category][word]
             else:
                count = 0
           self.prob[category][word] = (count + 1) / denominator
   def train(self, trainingdir, category):
        """counts word occurrences for a particular category"""
        currentdir = trainingdir + category
        files = os.listdir(currentdir)
        counts = {}
        total = 0
        for file in files:
           f = codecs.open(currentdir + '/' + file, 'r', 'iso8859-1')
           for line in f:
              tokens = line.split()
              for token in tokens:
                 # get rid of punctuation and lowercase token
                 token = token.strip('\'".,?:-')
                 token = token.lower()
                 if token != '' or not token in self.stopwords:
                    self.vocabulary.setdefault(token, 0)
                    self.vocabulary[token] += 1
                    counts.setdefault(token, 0)
                    counts[token] += 1
                    total += 1
           f.close()
        return(counts, total)
```

When we want to classify a file, we will call the classify method. This method will compute the probabilities that the file is in each category. Once we get rid of stop words, the probability of any specific word occurring is actually quite low. The occurrence of any particular word is a **rare event**. For example, say I want to categorize a document that starts: *The idea that unrecognized pantheism is dangerous to Christians*. First, I remove stopwords leaving: *idea unrecognized pantheism dangerous Christians*. Now I want to compute the probability that the document is from alt.atheism (here I abbreviate alt.atheism as *aa*):

P(*idea*|aa)		P(*unrecognized*|aa)		P(*pantheism*|aa)		P(*dangerous*|aa)		P(*Christians*|aa)
0.003	x	0.0000068086	x	0.0000068086		0.000081703	x	0.00034724

Multiplying these probabilities together gives me 3.94e-21 or

0.00000000000000000000394

If we multiple the word probabilities together for even a short document of 100 words we are going to get a very, very small number. Due to the limits of precision in Python, Python will just return 0 as the probability that that document belongs to a category. As a work around, instead of multiplying the probabilities, my classify code is going to add the logs of the probabilities.

```
def classify(self, filename):
    results = {}
    for category in self.categories:
       results[category] = 0
       f = codecs.open(filename, 'r', 'iso8859-1')
       for line in f:
          tokens = line.split()
          for token in tokens:
             token = token.strip('\'".,?:-').lower()
             if token in self.vocabulary:
               for category in self.categories:
                  results[category] += math.log(self.prob[category][token])
       f.close()
      results = list(results.items())
      results.sort(key=lambda tuple: tuple[1], reverse = True)
      # for debugging I can change this to give me the entire list
      return results[0][0]
```

Finally, I am going to add a few methods that will enable me to evaluate my classifier. It assumes my test files are arranged identically to that outlined above for the training files.

```
def testCategory(self, directory, category):
   files = os.listdir(directory)
   total = 0
   correct = 0
   for file in files:
      total += 1
      result = self.classify(directory + file)
      if result == category:
         correct += 1
   return (correct, total)


def test(self, testdir):
   """Test all files in the test directory--that directory is organized into
   subdirectories--each subdir is a classification category"""
   categories = os.listdir(testdir)
   #filter out files that are not directories
   categories= [filename for filename in categories if os.path.isdir(testdir + filename)]
   correct = 0
   total = 0
   for category in categories:
      (catCorrect, catTotal) = self.testCategory(testdir + category + '/', category)
      correct += catCorrect            total += catTotal
   print("Accuracy is  %f%%  (%i test instances)" % ((correct / total) * 100, total))
```

This code as well as the data, is available at http://guidetodatamining.com

Here is how the code works.

```
>>> c = BayesText('/Users/raz/Downloads/20news/divided/training/',
                  '/Users/raz/Downloads/20news/divided/stoplist.txt')
Counting ...
   alt.atheism
   comp.graphics
   …
   talk.religion.misc
Computing probabilities:
   alt.atheism
   comp.graphics
   …
   talk.religion.misc
>>> c.test('/Users/raz/Downloads/20news/divided/test/')
Accuracy is  86.569455%  (5212 test instances)
```

Given that we have 20 different newsgroups, random guessing would be about 5% accuracy. This algorithm, was 86% accurate. That's quite good.

Consider this bayes code used in a web app. A user would paste in some text in a web page, click 'submit' and our Python code would display what newsgroup it is from. Ok, the user pastes text & clicks submit. Our Python code is fired up. First, it goes through the over 10,000 training files to construct a classifier, which takes a bunch of time, and then it can quickly classify the text.

When we are experimenting with different classifiers, the approach taken in the above code is fine. For production environments, we would want to separate the training code and save the results. If we are using Python we might save the dictionary objects using shelf. When we want to classify, we load the probabilties instead of computing them from scratch.

Think of the possibilities!

Think of the possibilities of this technology—the ability to classify documents.

Custom news

Let's say you classify the news items you read as ones you like, and ones you dislike—just a little thumbs-up/thumbs-down button on your newsfeed. Eventually the system will learn what you like and display those articles in your customized 'newspaper'. This is similar to the app Flipboard for the iPad:

waltmossberg
Walt Mossberg
Hard Drives By Hitachi Back Up Files Two Ways | Walt Mossberg | Personal Technology | AllThingsD
Walt's Ethics Statement | AllThingsD

Nicholas Negroponte of M.I.T., calls this the "Daily Me."[1] Another example of this is MSNBC.com which has a feature that does text analysis on a person's most recently read articles to predict what news articles I would like to see. Professor Cass Sunstein of the University of Chicago Law School argues that "the Daily Me" is not a good thing and that it is beneficial for people to read shared common publications to expose people to a variety of viewpoints. Perhaps in the early 20th century people in a community were exposed to the same newspaper, and the same television nightly news. But, like it or not, those days of over. Personalized news is nothing new. What I read is different from what my neighbor reads, and what my wife reads. I subscribe to the *Rolling Stone Magazine*, *Keyboard*, *Wired*, and *Tricycle*. I scan certain websites daily: Hacker News, the Huffington Post,

1 Carl Kaplan "Law Professor Sees Hazard in Personalized News" New York Times April 13, 2001.

BoingBoing, LifeHacker. I do not subscribe to the local paper, the *Las Cruces Sun News* and I rarely check out their website. As a result I am pretty ignorant of what is happening locally. The Daily Me will take an evolutionary step forward. I don't read every article in the magazines I receive. If there is an article about Hiromi in Keyboard, I will read it. In the latest issue of Keyboard I skipped the article about Matt Rolling, Country's Keyboard King. If I was reading these online, we could use a naïve Bayes approach to analyzing the text of the articles I liked (in Rolling Stone, Keyboard, Wired, and Tricycle) and use the classifier to make the Daily Me for me. I would absolutely love it!

Other possibilities

Spam, Spam, and more Spam

You can use a Bayes approach to classify email as spam or not. You would have a training set already categorized as spam and not spam. A number of commercial spam filters are based on naïve Bayes methods.

Automatic essay grading

Tired of grading those essays? Let a program do it. A number of companies offer automatic essay grading software. Some of which are based on naïve Bayes methods. This seems hard to believe but research suggests they work quite well.

Author identification

Who really wrote that long terrorist manifesto posted on the web? Was it Alexander Hamilton or James Madison that wrote an essay published in 1788? Who wrote that mysterious love letter that you received yesterday? Naïve Bayes text analysis can at least help us answer the first two.

For all these applications (and many more) the approach is the same. First, you need a training set—a set of documents tagged with their correct classification. Hundreds of news articles I either read (liked) or skipped (not liked). One thousand emails some classified as Spam, some as not spam. Hundreds of documents written by Alexander Hamilton, and hundreds by James Madison. I train the naïve Bayes classifier (or classifier based on other methods) on this training set. I now have a classifier I can use to tag news items as those I would like and those I would not like; a classifier than scans my email for spam; a classifier to identify the author of a text.

Back to twitter and movies

Okay, I am going to crowd source this section to my first readers. I would like to see if we can categorize current movies based on twitter postings—was it a good movie or not? First, we need a training set – the larger the better consisting of postive tweets, negative tweets, and ignore tweets. For example:

- Christopher Nolan is such a GENIUS!!! I´ve seen all his movies and it´s all Great but INCEPTION is really something...such a Materpiece!
- Charlie St. Cloud was so good!
- Went to see Charlie St. Cloud last night. It was the cutest movie eveeeeer!
- In fact "cats and dogs" is so bad my 3yo and 5yo both asked to walk out of it, so we did!
- I saw inception and I think is great well the story and the photography great the actors were great
- Apparently I'm a 13yo girl, just saw Charlie St Cloud and didn't hate it.
- #inception was confusing and awesome! Great story amazing cast!
- Grown ups is a damn good film.
- Just saw Grown Ups,not all that great.
- Just enjoyed Inception. Im crying like a 12 year old girl! What an enjoyable film!.
- Inception was a fantastic movie

By 'ignore' tweets I mean anything that isn't a tweet reviewing the movie including:

- Im gonna see charlie st cloud today! :D
- Girls night me and sister going to watch Charlie St. Cloud
- About to see Charlie St. Cloud
- Dinner For Schmucks or Grown Ups?

So I would like to crowd source this task of collecting tweets. Once we have a good training set of tweets, I will post them to the website.

The next task will be to build a classify that can classify movie tweets as 'positive review', 'negative review' or other. Once we have that classifier, the question is how can we use this classifier to determine if a movie is good or not based on tweets.

Tools that may be useful in your explorations...

Mark Pilgrim's excellent Python book, Dive Into Python 3[2] has this to say about using Python to access web resources:

2 Available for free at http://www.diveintopython3.org and at bookstores including Amazon.

Python 3 comes with two different libraries for interacting with HTTP web services:

- http.client …
- urllib.request …

So which one should you use? Neither of them. Instead you should use httplib2 an open source third-party library that implements HTTP more fully than http.client, but provides a better abstraction than urllib.request.

Mark Pilgrim *Dive Into Python 3* ch. 14

He goes on to explain, in good detail, why we should use httplib2 and I recommend you do so. You can download the source code from http://code.google.com/p/httplib2/ as either a zip file or a gzipped tar file. Once you download it unzip it. So on Mac or Linux systems:

```
    tar xvzf httplib2-0.6.0.tar.gz
```

So, for example...

```
Ron-Zacharskis-iMac:Downloads raz$ tar xvzf httplib2-0.6.0.tar.gz
x httplib2-0.6.0/
x httplib2-0.6.0/MANIFEST
x httplib2-0.6.0/README
x httplib2-0.6.0/python2/
x httplib2-0.6.0/python2/httplib2test.py
x httplib2-0.6.0/python2/httplib2/
x httplib2-0.6.0/python2/httplib2/__init__.py
x httplib2-0.6.0/python2/httplib2/iri2uri.py
x httplib2-0.6.0/CHANGELOG
x httplib2-0.6.0/python3/
x httplib2-0.6.0/python3/README
x httplib2-0.6.0/python3/httplib2test.py
x httplib2-0.6.0/python3/httplib2/
x httplib2-0.6.0/python3/httplib2/__init__.py
x httplib2-0.6.0/python3/httplib2/iri2uri.py
x httplib2-0.6.0/setup.py
```

Once we do that we need to build and install the library

```
Ron-Zacharskis-iMac:Downloads raz$ cd httplib2-0.6.0
Ron-Zacharskis-iMac:httplib2-0.6.0 raz$ sudo python3 setup.py install
```

```
running install
running build
running build_py
creating build
creating build/lib
creating build/lib/httplib2
copying python3/httplib2/__init__.py -> build/lib/httplib2
copying python3/httplib2/iri2uri.py -> build/lib/httplib2
running install_lib
creating /Library/Frameworks/Python.framework/Versions/3.1/lib/python3.1/site-
packages/httplib2
copying build/lib/httplib2/__init__.py ->
/Library/Frameworks/Python.framework/Versions/3.1/lib/python3.1/site-packages/httplib2
copying build/lib/httplib2/iri2uri.py ->
/Library/Frameworks/Python.framework/Versions/3.1/lib/python3.1/site-packages/httplib2
byte-compiling /Library/Frameworks/Python.framework/Versions/3.1/lib/python3.1/site-
packages/httplib2/__init__.py to __init__.pyc
byte-compiling /Library/Frameworks/Python.framework/Versions/3.1/lib/python3.1/site-
packages/httplib2/iri2uri.py to iri2uri.pyc
running install_egg_info
Writing /Library/Frameworks/Python.framework/Versions/3.1/lib/python3.1/site-
packages/httplib2-0.6.0-py3.1.egg-info
Ron-Zacharskis-iMac:httplib2-0.6.0 raz$
```

An example of using httplib2 with twitter

Unfortunately, as of writing this chapter, the Twitter Python library was not available for Python 3. However, we can access the Twitter api using the httplib2 library. First, we need to create an instance of the httplib2.Http class:

```
>>> import httplib2
>>> h = httplib2.Http('.cache')
>>> response, content = h.request('http://search.twitter.com/search.json?q=love')
```

Here I retrieved tweeters with the word 'love' in them. If you look at the content of `content` you will notice it starts with a 'b' indicating bytes:

```
>>> content
b'{"results":
[{"profile_image_url":"http://a0.twimg.com/profile_images/1093100724/andreaj17_normal.jpg","
created_at":"Mon, 02 Aug 2010 17:53:45 +0000","from_user":"iAdmireTorres","metadata":
{"result_type":"recent"},"to_user_id":null,"text":"Spongebob: What kind of an old lady are
you? // I love spongebob.","id":2015661
…
```

Our next step is to convert the byte sequence to a character string:

```
>>> c = content.decode('utf8')
```

Looking at c we can see it is no longer a byte sequence (the obvious indicator is the absence of 'b' at the start of the sequence:

```
>>> c
'{"results":
[{"profile_image_url":"http://a0.twimg.com/profile_images/1093100724/andreaj17_normal.jpg","
created_at":"Mon, 02 Aug 2010 17:53:45 +0000","from_user":"iAdmireTorres","metadata":
{"result_type":"recent"},"to_user_id":null,"text":"Spongebob: What kind of an old lady are
you? // I love spongebob.","id":20156615
```

json

If you look at what I requested from twitter

```
>>> response, content = h.request('http://search.twitter.com/search.json?q=love')
```

you will notice that I used 'search.json' indicating that I am requesting the data be returned in the json format. Json stands for JavaScript Object Notion. It's a standard, light-weight, text-based, data interchange format. (I think I got all the required buzzwords in that.) It is an alternative to XML for exchanging data between servers and web application clients. Python has a built in library to handle json. One method in that library converts a json structure to a Python dictionary. That is what I will use here.

```
>>> import json
>>> x = json.loads(c)
>>>
```

Now, x is a standard Python dictionary and I can extract the results portion of the json structure.

```
>>> results = x['results']
>>> len(results)
15
```

It looks like I retrieved 15 twitters with my search query. Let's see the text of them:

```
>>> for twitter in results:
       print(twitter['text'])
Spongebob: What kind of an old lady are you? // I love spongebob.
yay thanks i have over 5,700 followers ! :) xx ánd i love them all !!
@Justinbieber we love you so so much! Will you notice us someday ;) ??
@JILLYPHONICS wednesday late afternoon early evening on the beach somewhere would love to
see you
RT @mllyssa @lojasbigolin @Colfr6n3m4n @Dirtydiamonds47 @Oldaa @Bebethsal @Sawda1
```

```
@Sonofcasanova Thx love
RT @MileysSmiler: RT this if you love us. :O
I'm listening to Visage. I love synthpop!
RT @MsEnergyHealer: &quot;Any real ecstasy is a sign you are moving in the right direction,
don't let any prude tell you otherwise.&quot; St. Therese (love this)
its all love getting the love that you love how i wrote about how i love
http://verdad.tumblr.com/post/892720958/on-the-way-out
@SnappMMI hehe....why is all of MMI taken over my damn timeline?? smh love u guys
RT @GARFIELDALLPRO: August 10th RT RT..If u got love 4 me make this easy soo I know the
Frenamiees... http://mypict.me/a2XzB
I can't be one day without say Bieber or I love Bieber♥
#ThatIsJustMe #nowplaying keri lynn-hilson &quot;make love&quot; we gonna make love love
love love love love love love love...let's MAKE LOVE :)
Saying &quot;I love burlesque dancing&quot; basicallly means &quot;I don't know that
pornography exists.&quot;
RT @sylvielesas: the weight of words/the value of feelings/your love forever #haiku
```

Alternatively, I can retrieve the users who posted these twitters:

```
>>> for twitter in results:
      print(twitter['from_user'])
iAdmireTorres
BieberPerfectx
WeFancyJBieber
benshephard
DirtyDiamonds47
xBabyBiebs
BimboBoy
danyasteele
Kate_Starr
Mzlollipop22
realmixtapemike
ThatIsJustMe
loveg89
jztdeathprof
Sahrazad528
```

The twitter api is fully explained at http://apiwiki.twitter.com/.

Hopefully, I have given you enough information about how to connect to Twitter with Python so that you will be able to retrieve twitters about specific movies

The challenge

The challenge is this: a movie opens on a Friday night, people, go see it and write Twitter posts about it: “Salt is fantastic”. Can we develop a system that can classify posts as being a postive review, a negative review, or not a review at all—the classifier can take a post about any movie and make this classification. Now the movie comes out on Friday, I grab the twitter posts about it for a few days—and I want to know whether the movie was a hit or not using the classifier plus a bit more code.

So that is the challenge.