**Sentiment analysis**

**Opinion mining** or **sentiment analysis** is a hot new research field dedicated to the automatic evaluation of opinions as expressed on social media, product review websites, or other forums. Often, we want to know whether an opinion is positive, neutral, or negative. This is, of course, a form of classification, as seen in the previous section. As such, we can apply any number of classification algorithms. Another approach is to semi-automatically (with some manual editing) compose a list of words with an associated numerical sentiment score (the word "good" can have a score of 5 and the word "bad" a score of -5). If we have such a list, we can look up all the words in a text document and, for example, sum up all the found sentiment scores. The number of classes can be more than three, as in a five-star rating scheme.

We will apply naive Bayes classification to the NLTK movie reviews corpus with the goal of classifying movie reviews as either positive or negative. First, we will load the corpus and filter out stopwords and punctuation. These steps will be omitted, since we have performed them before. You may consider more elaborate filtering schemes, but keep in mind that excessive filtering may hurt accuracy. Label the movie reviews documents using the categories() method:

labeled\_docs = [(list(movie\_reviews.words(fid)), cat)

for cat in movie\_reviews.categories()

for fid in movie\_reviews.fileids(cat)]

The complete corpus has tens of thousands of unique words that we can use as features. However, using all these words might be inefficient. Select the top 5 percent of the most frequent words:

words = FreqDist(filtered)

N = int(.05 \* len(words.keys()))

word\_features = words.keys()[:N]

For each document, we can extract features using a number of methods, including the following:

* Check whether the given document has a word or not
* Determine the number of occurrences of a word for a given document
* Normalize word counts so that the maximum normalized word count will be less than or equal to 1
* Take the logarithm of counts plus 1 (to avoid taking the logarithm of zero)
* Combine all the previous points into one metric

As the saying goes, *all roads lead to Rome*. Of course, some roads are safer and will bring you to Rome faster. Define the following function, which uses raw word counts as a metric:

def doc\_features(doc):

doc\_words = FreqDist(w for w in doc if not isStopWord(w))

features = {}

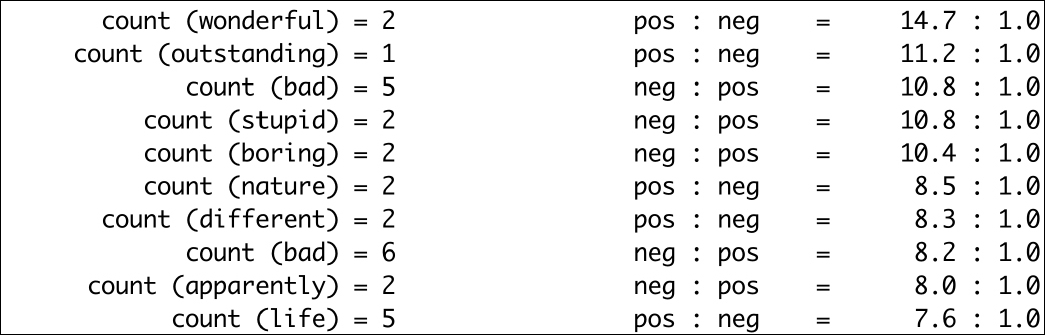
for word in word\_features:

features['count (%s)' % word] = (doc\_words.get(word, 0))

return features

We can now train our classifier just as we did in the previous example. An accuracy of 78 percent is reached, which is decent and comes close to what is possible with sentiment analysis. Research has found that even humans don't always agree on the sentiment of a given document (see <http://mashable.com/2010/04/19/sentiment-analysis/>), and therefore, we can't have a 100 percent perfect accuracy with sentiment analysissoftware.

The most informative features are printed as follows:



If we go through this list, we find obvious positive words such as "wonderful" and "outstanding". The words "bad", "stupid", and "boring" are the obvious negative words. It would be interesting to analyze the remaining features. This is left as an exercise for the reader. Refer to the sentiment.py file in this book's code bundle:

import random

from nltk.corpus import movie\_reviews

from nltk.corpus import stopwords

from nltk import FreqDist

from nltk import NaiveBayesClassifier

from nltk.classify import accuracy

import string

labeled\_docs = [(list(movie\_reviews.words(fid)), cat)

for cat in movie\_reviews.categories()

for fid in movie\_reviews.fileids(cat)]

random.seed(42)

random.shuffle(labeled\_docs)

review\_words = movie\_reviews.words()

print("# Review Words", len(review\_words))

sw = set(stopwords.words('english'))

punctuation = set(string.punctuation)

def isStopWord(word):

return word in sw or word in punctuation

filtered = [w.lower() for w in review\_words if not isStopWord(w.lower())]

print("# After filter", len(filtered))

words = FreqDist(filtered)

N = int(.05 \* len(words.keys()))

word\_features = words.keys()[:N]

def doc\_features(doc):

doc\_words = FreqDist(w for w in doc if not isStopWord(w))

features = {}

for word in word\_features:

features['count (%s)' % word] = (doc\_words.get(word, 0))

return features

featuresets = [(doc\_features(d), c) for (d,c) in labeled\_docs]

train\_set, test\_set = featuresets[200:], featuresets[:200]

classifier = NaiveBayesClassifier.train(train\_set)

print("Accuracy", accuracy(classifier, test\_set))

print(classifier.show\_most\_informative\_features())