In 2002 Paul Graham, having some time on his hands after selling Viaweb to Yahoo, wrote the essay “A Plan for Spam”\(^1\) that launched a minor revolution in spam-filtering technology. Prior to Graham’s article, most spam filters were written in terms of handcrafted rules: if a message has XXX in the subject, it’s probably a spam; if a message has a more than three or more words in a row in ALL CAPITAL LETTERS, it’s probably a spam. Graham spent several months trying to write such a rule-based filter before realizing it was fundamentally a soul-sucking task.

To recognize individual spam features you have to try to get into the mind of the spammer, and frankly I want to spend as little time inside the minds of spammers as possible.

To avoid having to think like a spammer, Graham decided to try distinguishing spam from nonspam, a.k.a. *ham*, based on statistics gathered about which words occur in which kinds of e-mails. The filter would keep track of how often specific words appear in both spam and ham messages and then use the frequencies associated with the words in a new message to compute a probability that it was either spam or ham. He called his approach *Bayesian* filtering after the statistical technique that he used to combine the individual word frequencies into an overall probability.\(^2\)

**The Heart of a Spam Filter**

In this chapter, you’ll implement the core of a spam-filtering engine. You won’t write a soup-to-nuts spam-filtering application; rather, you’ll focus on the functions for classifying new messages and training the filter.

This application is going to be large enough that it’s worth defining a new package to avoid name conflicts. For instance, in the source code you can download from this book’s Web site, I use the package name `COM.GIGAMONKEYS.SPAM`, defining a package that uses both the standard `COMMON-LISP` package and the `COM.GIGAMONKEYS.PATHNAMES` package from Chapter 15, like this:

---

1. Available at [http://www.paulgraham.com/spam.html](http://www.paulgraham.com/spam.html) and also in *Hackers & Painters: Big Ideas from the Computer Age* (O’Reilly, 2004)
2. There has since been some disagreement over whether the technique Graham described was actually “Bayesian.” However, the name has stuck and is well on its way to becoming a synonym for “statistical” when talking about spam filters.
(defpackage :com.gigamonkeys.spam
  (:use :common-lisp :com.gigamonkeys.pathnames))

Any file containing code for this application should start with this line:

(in-package :com.gigamonkeys.spam)

You can use the same package name or replace com.gigamonkeys with some domain you control.3

You can also type this same form at the REPL to switch to this package to test the functions you write. In SLIME this will change the prompt from CL-USER> to SPAM> like this:

CL-USER> (in-package :com.gigamonkeys.spam)
#<The COM.GIGAMONKEYS.SPAM package>
SPAM>

Once you have a package defined, you can start on the actual code. The main function you’ll need to implement has a simple job—take the text of a message as an argument and classify the message as spam, ham, or unsure. You can easily implement this basic function by defining it in terms of other functions that you’ll write in a moment.

(defun classify (text)
  (classification (score (extract-features text))))

Reading from the inside out, the first step in classifying a message is to extract features to pass to the score function. In score you’ll compute a value that can then be translated into one of three classifications—spam, ham, or unsure—by the function classification. Of the three functions, classification is the simplest. You can assume score will return a value near 1 if the message is a spam, near 0 if it’s a ham, and near .5 if it’s unclear.

Thus, you can implement classification like this:

(defun classification (score)
  (cond
    ((<= score *max-ham-score*) 'ham)
    ((>= score *min-spam-score*) 'spam)
    (t 'unsure)))

The extract-features function is almost as straightforward, though it requires a bit more code. For the moment, the features you’ll extract will be the words appearing in the text. For each word, you need to keep track of the number of times it has been seen in a spam and the number of times it has been seen in a ham. A convenient way to keep those pieces of data together with the word itself is to define a class, word-feature, with three slots.

3. It would, however, be poor form to distribute a version of this application using a package starting with com.gigamonkeys since you don’t control that domain.