

User's Guide for 'Baywatch'

Michael J. Fromberger
<sting@linguist.dartmouth.edu>

Version 1.3—September 10, 2002

This document describes how to use the `baywatch` Bayesian spam filtering program to help filter unsolicited commercial e-mail out of your mailbox. You can download the latest version of the program from <http://thayer.dartmouth.edu/sting/software.shtml>.

1 Requirements

In addition to the `baywatch` script itself, you will need:

1. A copy of the Perl 5 interpreter, including the standard modules `DB_File`, `Fcntl`, and `Getopt::Long`.
2. A collection of messages you consider to be spam, in Unix mailbox format. We will call this the “spam corpus.”
3. A collection of messages you consider to be good (not spam), also in Unix mailbox format. We will call this the “good corpus.”

If necessary, edit the first line of the `baywatch` script to invoke the correct version of the Perl interpreter, make the script itself executable, and save it somewhere in your path.

2 Getting Started

This section describes how to get `baywatch` set up for the first time, and how to use it to test new messages that arrive.

2.1 First-Time Setup

Before `baywatch` will be useful as a spam filter, you need to provide it with some training data. This consists of some messages you have deemed to be spam, and some messages you have deemed to be good. These messages are assumed to be saved in the standard Unix mailbox format (with `'From'` lines indicating the start of a new message).

The program will extract the words from these messages and create a database of statistics that it can later use to guess the probability that a new message (hitherto unseen) is spam.¹

First, you need to direct `baywatch` to load in all your training messages. The more messages you can provide up front, the more accurate your results are likely to be. Loading the training messages lets `baywatch` create a database of word statistics, which is used to test new messages.

¹For a detailed description of how this probability is computed, read “Bayesian Classification of Unsolicited E-Mail,” the white paper distributed with this program.

For a file `goodfile` containing good messages, run:

```
baywatch --good goodfile
```

For a file `spamfile` containing spam examples, run:

```
baywatch --spam spamfile
```

These options may be combined to load both files simultaneously, and may be repeated to load multiple files, e.g.:

```
baywatch --good good1 --spam spam1 --spam spam2 --good good2
```

You may load as many files as you wish, and additional files may be loaded at any time using the same procedure.

2.2 Testing New Messages

Once you have loaded your training messages and created the word frequencies table, you can direct `baywatch` to test a new message for the likelihood that it is spam. You do this using the `--test` command line option. For example, if you have a new message in a file named `message.txt`, you can test it by running:

```
baywatch --test message.txt
```

The message will be analyzed, and the probability that it is a spam message will be printed to the standard output. Probabilities are given as decimal numbers between 0 and 1, e.g.:

```
% baywatch --test message.txt
0.012725
```

If you do not provide an input file name, `baywatch` reads from its standard input.

For filtering purposes, you don't usually want `baywatch` to print out the probability, but instead to exit with some non-zero status if the probability is high enough to consider the message to be spam. To do this, use the `--return` command line option, e.g.:

```
% baywatch --test --return 0.95 message.txt && echo "Message is OK"
Message is OK
```

If the computed probability is less than or equal to the specified threshold value, `baywatch` returns an exit status of 0; otherwise, it exits with a status of 1. The numeric argument of `--return` may be omitted, in which case it defaults to 0.9.

You can also run `baywatch` interactively, and get it to display the criteria it used for judging the message. To do this, use the `--display` option. In addition to printing out the probability of the message being spam, it will also summarize some parameters of operation, and the spam probabilities of individual words. See Figure 1 below for an example.

For most filtering jobs, you will want to pipe the incoming message to `'baywatch --test --return'`, which will silently test the message and indicate its results as an exit status.

```

Interest threshold: 15
Interest bandwidth: 0.090
Probability inset: 0.010
Frequency scaling: no
Novelty bias:      0.400

      href 0.9965
      click 0.9953
      arial 0.9946
      font 0.9927
      gif 0.9907
      verdana 0.9900
      alt 0.9900
marginheight 0.9900
      length 0.9900
      bureaus 0.9900
      offers 0.9900
      ffffff 0.9900
      border 0.9900
      guaranteed* 0.9900
      platinum 0.9900
1.000000

```

Figure 1: Output of the ‘display’ option

3 Integration with Procmail

To integrate `baywatch` with the `procmail` mail delivery agent, you will need to write a recipe that uses `baywatch` to test incoming messages, and then chooses where to deliver them based on its return value. Here is a recipe to do the testing:

```

:0 Wc
|baywatch --test --return

```

Because of the ‘c’ flag, this is not a delivering recipe; it just pipes the message to `baywatch` in order to get its analysis back, and waits (‘W’) for the result. Now, you can choose to take a special action based on the results. A simple possibility is to just file the message in a spam folder, *e.g.*:

```

:0 e:
spamfolder

```

The ‘e’ flag causes this recipe to be run only if the exit status of the previous recipe indicated failure. Thus, this recipe should be put right after the previous one, in your recipes file.

Alternately, you could opt for more interesting behaviour. This next recipe uses `formail` to add an `X-Spam:` header to the message, and then save it in your spam folder. Again, as in the previous example, we’re using the ‘e’ flag to insure that this recipe is only run if the previous one (the one giving the message to `baywatch` for analysis) has failed:

```

:0 e
{
  :0 fhw
  |formail -I "X-Spam: Detected by baywatch"

  :0:
  spamfolder
}

```

You could also run `baywatch` without the ‘`--return`’ option, and instead capture the probability value and use it in conjunction with various “scoring” features of certain mail clients (e.g. `mutt`) to do client-side sorting instead of delivery-side.

Note: In order for `baywatch` to run correctly unattended, it is necessary that your words database be located somewhere it can be read by the system while delivering e-mail. In most environments, this is simple, since the mail delivery agent runs either as the user for whom mail is being delivered, or as a privileged user. However, on some systems—notably AFS—that is not enough by itself.

The ‘`--test`’ option requires only read access to the database, but the training options (e.g., `--spam` and `--good`) will need both read and write access.

4 Reference

This section describes all the options understood by `baywatch`.

- `--good <path>`
Add a file of “good” training examples to the database.
- `--spam <path>`
Add a file of spam training examples to the database.
- `--unlearn`
Remove messages from the database. Used in conjunction with the `--spam` and `--good` options.
- `--exchange`
Reclassify good messages as spam or spam messages as good. Used in conjunction with the `--good` and `--spam` options. Equivalent to using `--unlearn` and then reloading the messages with the opposite classification.
- `--flush`
Causes the database to be flushed, and all word frequencies deleted. Use this if you want to reset the database before loading an entirely new set of training data.
- `--dbfile <path>`
Use the specified file as the word frequencies database. The default is `.worddb` in the user’s home directory.
- `--dbexport <path>`
Export the word frequencies database as a flat text file. The resulting file can be turned back into a normal database using the `--dbimport` flag. This is useful when going from one platform to another, where the binary databases have different formats.
- `--dbimport <path>`
Import word frequencies from the flat text format generated by `--dbexport`, and replace the data in the current word frequencies database with it.

--bias <prob>
 Sets the default spam probability for unknown words that occur in a message given for testing. Defaults to 0.4, which is slightly forgiving.

--interest <n>
 Sets the number of interesting words that will be considered when estimating the spam probability of a new message. The default is 15.

--width <prob>
 In order to be considered interesting, a word's spam probability must differ from 0.5 by at least this much. The default value is 0.09, and may be adjusted by setting this option.

--return [<prob>]
 Instead of printing out the computed probability to standard output (which is the default behaviour), the program's exit status will indicate whether the tested message is spam or not. If the computed probability is greater than the given value, the message is spam and the exit status is 1. Otherwise, the exit status is 0. If omitted, a probability of 0.9 is assumed.

--crap <level>
 Set the level of novelty a word must contain in order to be considered (see Section 4.2).

--normalize
 Instead of computing spam probabilities based on individual word frequencies, normalize all frequencies by dividing through by the number of messages over which those frequencies were computed. This changes the emphasis in some cases.

--plain
 Treat the input as plain text, rather than as an e-mail message. Normally, mail headers are parsed out and discarded; with this flag, they are left in and processed as if they were part of the message text.

--words [<word>+]
 Print out word frequency statistic and spam probabilities for the given words.

--filter <expr>
 When combined with the '**--words**' option, displays only the words matching the particular expression you describe. See section 4.1 for a synopsis of the expression language understood.

--delete
 When combined with the '**--words**' option, all the words selected will be deleted from the database, in addition to being listed. You can combine this with '**--filter**' to get rid of words that match certain criteria.

--display
 When testing a message, print out interesting statistics about the message along with the probability.

--log
 When testing a message, print out the computed probability and other information for logging purposes, to standard output. Not as comprehensive an output as the **--display** option provides.

--test
 Test a message to see if it is spam.

4.1 Filter Expressions

The '**--filter**' option allows you to specify a simple Boolean description of various word attributes. The attributes of a word are:

f	The frequency of the word in the training corpus
fs	The frequency of the word in spam messages
fg	The frequency of the word in good messages
v	The average number of occurrences per message (total)
vs	The average number of occurrences per spam message
vg	The average number of occurrences per good message
p	The spam probability of the word (unnormalized)
ps	Same as p
pn	The normalized spam probability of the word
c	The “crap index” of the word
len	The length of the word, in characters
key	The word itself
px	The prefix of the word
sfx	The suffix of the word

For all attributes except **key**, **px**, and **sfx**, the comparison operations permitted are:

=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to

For frequencies (**f**, **fs**, and **fg**), the comparison values are decimal integers. For instance, to match all words which occur at least 100 times in the corpus, you would write the expression **f >= 100**.

Probabilities (**p**, **ps**, and **pn**), a non-negative decimal fraction less than 1 is required. So, to match all words whose spam probabilities are less than 0.2, you would write **ps < 0.2** or **ps < .2**.

Averages are written as arbitrary decimal values. So, to find any word which occurs in good messages an average of 5.1 times, you would write **vg = 5.1**.

The **key**, **px**, and **sfx** attributes can only be compared using = (equals), and their comparison values are words. So, to find any word beginning with “the,” you would write **px = the**. Word comparisons are done case-insensitively.

Compound queries can be constructed using the Boolean **&** (AND) and **|** (OR) operators. They associate to the left, with AND having higher precedence than OR. You may enforce a different order using parentheses in the usual way. For instance, to find all words of at least ten characters whose spam probability is more than 0.2 away from 0.5, you could write **len >= 10 & (p < .3 | p > .7)**.

Whitespace is ignored within expressions, so the latter could have been written **len>=10&(p<.3|p>.7)** with equivalent meaning.

4.1.1 Filter Expression Errors

The following errors may be reported when interpreting your filter expressions:

Invalid input in filter expression

This means you gave an invalid expression of some kind. This might happen if you enter the wrong kind of comparison value for a word attribute (a decimal fraction for a frequency, for instance).

Syntax error in filter expression

This means that the syntax of your filter expression was wrong in some way, such as an unbalanced parenthesis or two adjacent comparisons without a connecting operator.

Missing operand in filter expression

This happens when you leave out one of the operands of an AND or OR operator.

Unexpected junk in filter expression

This means when we got to the end of your filter expression, there was something there we didn't expect to see.

Nothing in filter expression

This means you provided an empty filter expression.

4.2 How Words are Extracted

One important question is how `baywatch` identifies what constitutes a “word” in the text you ask it to analyze. Right now, the following rules determine what constitutes a word:

1. The input is broken into a series of maximal-length consecutive runs of letters (A-Z), decimal digits (0-9), apostrophes, dollar-signs (\$), and asterisks (*). Letters are considered without regard to capitalization.
2. Any such sequence containing fewer than 3 characters or more than 25 characters is discarded.
3. Any remaining sequence which consists only of digits is discarded.
4. Any remaining sequence which does not contain at least one letter (A-Z) is discarded.
5. Any remaining sequence whose “crap ratio” is greater than 0.7 is discarded.
6. The remaining sequences are considered words.

The so-called *crap ratio* is a measure of how likely the word is to be a “real” word, as opposed to something masquerading as a word such as a Base64 encoded MIME attachment. It is computed by dividing the number of unique characters in the word by the total length of the word in characters, and subtracting this value from 1. A very high ratio is uncommon among real words in a natural human language.

I chose 0.7 as a cutoff by computing the crap ratio of every word in `/usr/share/dict/words` on my system, and selecting a value close to the largest ratio found. Here are some statistics on crap ratios computed over the words in my system dictionary:

Total number of words	234,937
Minimum ratio	0.000
First quartile	0.111
Median	0.182
Third quartile	0.273
Maximum	0.714
Average (mean)	0.187
Variance	0.015
Standard deviation	0.123

Indeed, only 478 of these words (around 0.2%) have ratios greater than 0.5, so it looks like at least for English words, this ratio is not a bad metric for whether a sequence is really a word.

On the other hand, “crap” may turn out to be a good predictor for spam, so discarding these words might not be as useful as one might hope. Still, as a means for keeping the size of your state table down, it is worthwhile to get rid of sequences that aren’t at least somewhat linguistic in nature.