# BayesFeedbackViaForwarding

## Feeding back mail for the Bayesian learner via forwarded mail

This is a form of SiteWideBayesFeedback.

For MUAs (Like Netscape/Mozilla) that do a good job with keeping orignal headers intact, (almost) all you need to do is forward the email to the feedback account and strip off the header added by the forward ( provided that you forward inline. I'll try to update bayes_fixup.pl for forwarding as attachment at a later date). This can be done by calling a filter from the ~/.procmailrc file of the learner accounts. (I apologize for putting these scripts in the Wiki, but I have no publically accessable location to post them, If someone who does has that capability, and could just replace them with links, I'd appreciate it)

I am not sure how sa-learn will interpret a signature when you forward email inline, so you should probably delete your .sig before forwarding the message.

I call spamc from /etc/procmailrc, but I make sure that it does't filter mail to is_spam and not_spam

/etc/procmailrc

```
    # Don't filter mail to is_spam and not_spam
    #
    #       Since we are running sitewide, it could cause a serious bottleneck if
    #       we were to use a lockfile here.  instead, we limit spamd to 20 child
    #       processes in /etc/sysconfig/spamassassin
    #
    #:0fw: spamassassin.lock
    :0fw
    * !^To.*spam@mycompany.com
    * < 256000
    | spamc
```

~is_spam/.procmailc ( I also have a ~not_spam/.procmailrc that is identical )

```
    # filter spam feedback
    :0fw: bayes_fixup.lock
    * < 256000
    | /usr/local/adm/bin/bayes_fixup.pl
```

bayes_fixup.pl is:

```
#!/usr/bin/perl
#
#       This filter is designed to pull off the forwarding headers for mail
#       forwarded to is_spam or not_spam from an MUA that includes all
#       headers.  ( as opposed to outlook, which does not include all
#       headers, and thus must be resent instaed of forwarded. )
#
#       In a forwarded message from Netscape/Mozilla, you will have:
#
#               From ...
#               ...
#               From: (matches envelope from)
#               ...
#               one or more blank lines
#               -------- Original Message --------
#               From: (a date code for the forwading MUA)
#               The original Headers
#
#       You will not have:
#               Sender:
#
#       Not sure if the Netscape stuff is valid for HTML mode.
#
#       Brian R. Jones  01/30/04 scumpuppy_@_earthlink_._net
#
use strict;

my ($count,$endheader,$sender,$unknown);
my $fwdmarker = "-------- Original Message --------";
```

```
my @message = <STDIN>;

#
#       Determine if sender is Outlook, Netscape/Mozilla or unknown.
#       If Netscape, set a marker for the end of the headers that are added
#       by the forwarding.
#
for( $count = 0, $endheader = 0, $sender = 0, $unknown = 0; ; $count++ ) {
        $_ = $message[$count];
        /^Sender:/o and last;     # It's a resent message from Outlook, skip
        /^\s*$/o and do {         # end of headers marked with one or more
                $endheader = 1;   # blank lines
                next;
        };
        next unless $endheader;
        /^$fwdmarker/o or $unknown = 1;
        last;
}
#
#       If it's Netscape, delete the forwarding header, and clean up the
#       original. I'm also converting the 'From:' to the 'Envelope From'
#       which may not be legitimate.  It may be better to use the forward
#       header 'Envelope From'.  Unfortunately, there is no way to capture
#       the original 'Envelope From'.  :(
#
if ( $endheader && ! $unknown ) {           # forwarded from known mailer
        splice(@message, 0 , ++$count);
        $message[0] =~ s/^From:/From/;
        for ( @message ) {                  # Stupid Netscape collapse continuation lines,
                                            # so we need to put `em back in case sa-learn
                                            # doesn't understand `em.
                /^[\w\-]+:/     and next;   # Valid header
                /^\t/           and next;   # Valid Continuation line
                /^From/         and next;   # Newly created Envelope From
                /^\s*$/         and last;   # End of Headers
                $_ = "\t" . $_;             # Malformed continuation line. Add tab.
        }
} elsif ( $unknown ) {          # unknown, toss it.
        exit 1;
}
print @message;
```

So all of the above handles delivery of a nearly (except for the 'envelope From') untainted message to the spam (is_spam) and ham (not_spam) accounts on the server. Note that these messages live where sendmail sends them. Next you need to run sa-learn on them, and sa-learn requires they first be split into individual messages. To do that, I call another script (learn_spam.pl) from cron. Since I'm using a Redhat Linux box I do it like this:

/etc/cron.daily/learnspam ( When you are testing, remove the redirect to /dev/null and cron will automatically email you (assuming you are root) the output from learn_spam.pl):

```
#!/bin/bash
#
#       run sa-learn on mail sent to the is_spam and not_spam accounts
#
/usr/local/adm/sbin/learn_spam.pl > /dev/null
```

/usr/local/adm/sbin/learn_spam.pl:

```
#!/usr/bin/perl -I/usr/local/lib
#
#       run sa-learn on is_spam and not_spam to update spamassassin
#
#       brj     01/27/04

use strict;
use Cwd;
require "splitmail.pl";


my $spamfile    = "/var/mail/is_spam";
my $hamfile     = "/var/mail/not_spam";
my $tmpdir      = "/var/tmp/split";

#my $learn_spam = "sa-learn --spam --showdots --dir $tmpdir";
#my $learn_ham  = "sa-learn --ham --showdots --dir $tmpdir";

my $learn_spam  = "sa-learn --spam --dir $tmpdir";
my $learn_ham   = "sa-learn --ham --dir $tmpdir";

my $startdir = cwd();

sub init {
        if ( ! -d $tmpdir ) {
                mkdir $tmpdir;
        } else {
                if ( chdir($tmpdir) ) {
                        unlink <*>;
                }
                chdir($startdir);
        }
}

sub learn {
        my $infile  = shift;
        my $command = shift;
        if ( -r $infile ) {
                splitmail($infile,$tmpdir);
                system("$command");
                if ( chdir($tmpdir) ) {
                        unlink <*>;
                }
                chdir($startdir);
        }
}

sub cleanup {
        unlink $spamfile, $hamfile;
        rmdir $tmpdir;
}

init();
learn( $spamfile, $learn_spam );
learn( $hamfile,  $learn_ham  );
cleanup();
```

Since I have several other apps that also require splitting a mail file I wrote splitmail (or maybe I borrowed it from someone else, I'm not sure) as a library.

/usr/local/lib/splitmail.pl:

```
#!/usr/bin/perl
#
#       splits a file containing multiple messages into individual files
#
use strict;

sub splitmail {
        my $infile = shift;
        my $outdir = shift;
        my $count = 0;

        open(INFILE, "< $infile") or die "Can't open $infile: $!\n";

        while(<INFILE>) {
                /^From / and do {
                        close(OUTFILE) if $count;
                        open(OUTFILE, "> $outdir/$count") or die "Can't open $outdir/$count: $!\n";
                        $count++;
                };
                print OUTFILE $_;
        }
        close(OUTFILE);
}

1;
```

Alternately, you can use this wrapper for sa-learn and call it from a .qmail file for on-the-fly split-and-learn-via-forward.

/usr/bin/learn_spam:

```
#!/usr/bin/perl
#
#      run sa-learn on STDIN ... easy to use with .qmail files:
#
#  .qmail-spamtrap:
#      | learn_spam --spam --username=alias | cat - > /dev/null
#  .qmail-qqqhamreport:
#      | learn_spam --ham --username=alias | cat - > /dev/null
#
#  3/16/2005 -- cgg007 at yahoo.com
#

use strict;

sub learn {
  my $message = shift;
  my $pipe = shift;
  open LEARN, $pipe;
  print LEARN $message;
  close LEARN;
}

my $learn_cmd =  "| bayes_fixup.pl | sa-learn " . join(" ",@ARGV);
my $count = 0;
my $message = '';

while (<STDIN>) {

  /^From/ and do {
    if ($count) {
      learn($message,$learn_cmd);
      $message = '';
    }
    $count++;
  };

  $message .= $_;

}

learn($message,$learn_cmd);
```