RunningGa

Running the GA to generate scores

As used in the RescoreMassCheck process.

Firstly, check that the rules and logs are both relatively clean and ready to use.

Copy/link the full source logs to "ham-full.log" and "spam-full.log" in the masses directory. Then:

```
cd masses

make clean
rm -rf ORIG NSBASE SPBASE ham-validate.log spam-validate.log ham.log spam.log
svn revert ../rules/50_scores.cf
ln -s ham-full.log ham.log
ln -s spam-full.log spam.log
make freqs SCORESET=3
less freqs
```

Go through the HitFrequencies report in freqs and check:

- ALL_TRUSTED hitrate on spam. This should appear only in ham.
- unfamiliar rules with high ham hitrates; they could be easily forgeable. comment them or mark them "tflags nopublish".
- NO_RECEIVED hitrate in spam.
- NO_RELAYS hitrate in spam.

Save a copy of freqs, then generate ranges:

```
cp freqs freqs.full
make > make.out 2>&1
less tmp/ranges.data
```

examine tmp/ranges.data and check:

- ranges that are 0.000 0.000 0 for no obvious reason;
- rules named with a "T_" prefix. These can sometimes slip through if used in promoted meta rules. They should be fixed to not include a "T_" prefix in the rulesrc source file. (that should be the only way that a T_ rule will appear in the output; "real" sandbox T_ rules should be removed already, since you deleted the sandbox rule file.)

To prepare your environment for running the rescorer:

```
  rm -rf ORIG NSBASE SPBASE ham-validate.log spam-validate.log ham.log spam.log
  mkdir ORIG
  for CLASS in ham spam ; do
  ln $CLASS-full.log ORIG/$CLASS.log
  for I in 0 1 2 3 ; do
  ln -s $CLASS.log ORIG/$CLASS-set$I.log
  done
  done
```

Score generation

Copy a config file from "config.set0"/"set1"/"set2"/"set3" to "config", and execute the runGA script. runGA generates and uses a randomly selected corpus with 90% being used for training and 10% being used for testing.

You need to ensure an up-to-date version of perl is used. On the zone, this is /local/perl586.

```
export PATH=/local/perl586/bin:$PATH
nohup bash runGA &
tail -f nohup.out
```

monitor progress... once the GA is compiled, and starts running, if the FP%/FN% rates are too crappy, it may be worth CTRL-C'ing the runGA process and running a new one "by hand" with different switches:

```
./garescorer -b 5.0 -s 100 -t 5.0
```

if you do this though you will have to cut and paste the post-GA commands (in the "POST-GA COMMANDS" section of runGA) by hand!

Once the GA run is complete, and you're happy with the accuracy: You will find your results in a directory of the form "gen-$NAME-$HAM_PREFERENCE-$THRESHOLD-$EPOCHS-$NOTE-ga".

Compare the listed FP%/FN% rate on gen-*/test to gen-*/scores; gen-*/scores is the output from the perceptron, and should match within a few 0.1% to gen-*/test output (which is computed on a separate subset of the mail messages). This checks:

- that the mail messages are diverse enough to avoid overfitting (hence the different test and train sets)
- that the FP%/FN% computations are not losing precision due to C-vs-Perl floating-point bugs, or a differing idea of what rules are promoted vs not promoted between the C and Perl code.

Once you're satisfied, check in ../rules/50_scores.cf. Copy the "config" file back to "config.setN" where "N" is the current scoreset, and check that in. Then, add a comment to the rescoring bugzilla bug, noting:

- the "gen-*/test" file contents, with FP%/FN% rate
- the "gen-*" path for later reference

next, carry on with other steps from RescoreMassCheck (if that's what you're doing).

PGAPack

To get garescorer to build with the above "make > make.out 2>&1" command on an Ubuntu Maverick machine, I installed the libpgapack-serial1 package, and ran:

```
mkdir -p /local/pgapack-1.0.0.1/lib
ln -s /usr/lib /local/pgapack-1.0.0.1/lib/sun4
mkdir -p /local/pgapack-1.0.0.1
ln -s /usr/include/pgapack-serial /local/pgapack-1.0.0.1/include
```

The first symptom you are likely to see of this problem is the error:

```
time: cannot run ./garescorer: No such file or directory
```

To take advantage of multiple CPU cores, use pgapack-mpi (which appears to be broken on ubuntu), and run it as:

```
mpirun -np 4 ./garescorer -b 10 -e 5500 -t 5.0
```

Replace "4" with your number of CPU cores. Although it looks like this causes redundant processing instead of distributed load.