MassesOverview

Tools in the SpamAssassin masses folder

The masses folder was included in the source tarball prior to SA 3.2.0. Since few users actually use these tools, they have been dropped from the distribution. If you have need of them, they can be publicly fetched from SVN.

This is an overview of the scripts in the SpamAssassin masses folder. In brief these scripts are used to mass check hand classified corpora and to calculate new scores with the percpetron approach using the results of a mass check. It's necessary to calculate 4 different scoresets for the rules, depending on whether the bayes or the net option is used:

set0: no bayes, no net

set1: no bayes, net

set2: bayes, no net

set3: bayes, net

A scoreset is one of the 4 columns in a score file like "../rules/50_scores.cf"

cpucount

This script counts the number of CPU in your system

usage:

cpucount

cpucount calls:

no other scripts

fp-fn-statistics

Tests a scoreset and *.log files for false-positives and false-negatives and returns a statistic.

usage:

fp-fn-statistics [options]

--cffile=file	path to *.cf files. Defalut: "../rules"
--lambda=value	lambda value, default: 50
--threshold=value	mails above the threshold are classified as spam
--spam=file	spam logfile, default: "spam.log"
--ham=file	ham logfile, default: "ham.log"
--scoreset=value	scoreset (0-3), default: 0
--fplog=file	false-positives logfile (list of false negatives)
--fnlog=file	false-negatives logfile (list of false positives)

fp-fn-statistics calls:

logs-to-c with --count option

hit-frequencies

see HitFrequencies.

hit-frequencies calls:

parse-rules-for-masses

lint-rules-from-freqs

This script analyzes the rules for usability. It therefore uses a freqs file generated by hit-frequencies (with -x -p options). It also uses a scoreset. The bad rules are returned. Following rules are marked as bad:

Rules that rarely hit (below 0.03%) or don't hit at all, rules with a negative score that have a higher spam-hit rate than ham-hit rate, rules with a positive score that have a higher ham-hit rate than spam-hit rate, rules with score = 0.

usage:

lint-rules-from-freqs [-f falsefreqs] [-s scoreset] < freqs > badtests

-f falsefreqs	also use a "falsfreqs" file for the analysis that was generated with hit-frequencies and -x -p -f options.
-s scoreset	scoreset (0-3).

lint-rules-from-freqs calls:

no other scripts

logs-to-c

Generates different files in the /tmp folder: "ranges.data", "scores.data", "scores.h", "tests.data", "tests.h". Those files are later used by the perceptron script. This script is also used to test scoresets and *.log files for false-positives and false-negatives (use --count).

usage:

logs-to-c [options]

--cffile=file	path to *.cf files. Defalut: "../rules"
--count	create fp-fn statistic
--lambda=value	lambda value, default: 50
--threshold=value	mails above the threshold are classified as spam
--spam=file	spam logfile, default: "spam.log"
--ham=file	ham logfile, default: "ham.log"
--scoreset=value	scoreset (0-3), default: 0
--fplog=file	false-positives logfile (list of false negatives)
--fnlog=file	false-negatives logfile (list of false positives)

logs-to-c calls :

parse-rules-for-masses

score-ranges-from-freqs

mass-check

see MassCheck.

mass-check calls:

no other scripts in the masses folder

mk-baseline-results

Shell script that tests a scoreset and the files "ham-test.log" and "spam-test.log" for false-positives and false-negatives with various thresholds ranging from -4 up to 20. Returns a statistic for all thresholds.

usage:

mk-baseline-results scoreset

scoreset	desired scoreset (0-3)

mk-baseline-results calls:

logs-to-c

parse-rules-for-masses

Parses the rules in all *.cf files that begin with a digit and that are located in the "../rules" folder.It generates a file called "/tmp/rules.pl" that contains a dump of two hashes (perl datatype) called %rules and %scores that can be directly included by other perl scripts using the require command.

The %rules hash consists of a set of data for every rule. In those sets, the score of the rule, a description, the type, whether the rule is mutable and whether it is a subrule are saved. In the %scores hash one score for every rule is saved.

usage:

parse-rules-for-masses [-d rulesdir] [-o outputfile] [-s scoreset]

-d	directory of the rules, default: ../rules
-o	output file, default: ./tmp/rules.pl
-s	scoreset (0-3), default: 0

parse-rules-for-masses calls:

no other scripts

perceptron

Calculates new scores with the perceptron approach and generates a perceptron.scores file. Needs following files in the /tmp folder: "ranges.data", "scores.data", "scores.h", "tests.data", "tests.h", "rules.pl"

usage:

perceptron [options]

-p ham_preference	adds extra ham to training set multiplied by number of tests hit (2.0 default)
-e num_epochs	number of epochs to train (15 default)
-l learning_rate	learning rate for gradient descent (2.0 default)
-t threshold	minimum threshold for spam (5.0 default)
-w weight_decay	per-epoch decay of learned weight and bias (1.0 default)
-h	print help

perceptron calls:

no other scripts

rewrite-cf-with-new-scores

Rewrites a cf file with new scores. Only the area with the generated scores is changed. The argument scoreset is the number of the scoreset (0-3) that is rewritten. The new cf-file is returned on the standard output.

usage:

rewrite-cf-with-new-scores [scoreset] [oldscores.cf] [newsocres.cf]

scoreset	desired scoreset to write (0-3)
oldscores.cf	old scores
newscores.cf	new scores

rewrite-cf-with-new-scores calls:

no other scripts

runGA

Shell script that compiles and runs the perceptron script. New scores are calculated with the perceptron approach and random 9/10 of the examples in the "*.log" files. Then the scores are tested for false-positives and false-negatives with the last 1/10 of the examples.

Needs a "config" file in the "./" folder that contains some parameters:

SCORESET=value	number of the scoreset (0-3)
HAM_PREFERENCE=value	ham preference for the perceptron
THRESHOLD=value	minimum threshold for spam
EPOCHS=value	number of epochs to train the perceptron

Corresponding "*.log" files to the chosen scoreset X (named "ham-setX.log" and "spam-setX.log") are required in the "/ORIG" folder. The script generates several files in the "/tmp" folder by calling logs-to-c, and a new folder named by the options ("gen*") in the config file. This folder contains a "scores" file with the generated scores and corresponding ranges, the "*.log" files that were used for the score generation and for the testing (in "/NSBASE" and "/SPBASE" folders), lists of false-negatives and false-positives that were found in the test, a logfile that contains the used parameters for the score generation, the output of the makefile ("make.output") and a false-positives vs. false-negatives statistic ("test").

The runGA script also generates a "badrules" file by calling lint-rules-from-freqs, that contains rules that are not useful for different reasons (most of them hitting too rarely or not at all).

Note that the generated scores may vary somewhat if runGA is run twice, due to the random selection of the training examples.

usage:

runGA (parameters are saved in a "config" file)

runGA calls:

fp-fn-statistics

lint-rules-from-freqs

logs-to-c

mk-baseline-results

numcpus

parse-rules-for-masses

perceptron

rewrite-cf-with-new-scores

score-ranges-from-freqs

tenpass/split-log-into-buckets-random

score-ranges-form-freqs

Calculates a score-range for the rules. The magnitude of the range depends on the ranking (generated by hit-frequencies) of a rule. Immutable rules get fixed ranges at their scores. The ranges are later used by the perceptron script that tries to find the optimal scores within these ranges.

usage:

score-ranges-from-freqs [cffiledir] [scoreset] < freqs

cffiledir	directory of the rules, default: "../rules"
scoreset	desired scoreset(0-3)

score-ranges-from-freqs calls:

parse-rules-for-masses

split-log-into-buckets-random

Split a mass-check log into n identically-
sized buckets, evenly taking messages from all checked corpora and preserving comments. Creates n files named "split-n.log"

usage:

split-log-into-buckets-random [n] < LOGFILE

n	number of buckets, default: 10

split-log-into-buckets-random calls:

no other scripts