CRM114 versus Mr. X:

CRM114 Notes for the TREC 2005 Spam Track

Fidelis Assis[1], William Yerazunis[2], Christian Siefkes[3], Shalendra Chhabra[2,4]

1: Empresa Brasileira de Telecomunicações – Embratel, Rio de Janeiro, RJ, Brazil

2: Mitsubishi Electric Research Laboratories, Cambridge MA

3: Database and Information Systems Group, Freie Universität Berlin, Berlin-Brandenburg Graduate School in Distributed Information Systems

4: Computer Science and Engineering, University of California, Riverside CA

Abstract: This paper discusses the design decisions underlying the CRM114 Discriminator software, how it can be configured as a spam filter, and what we may glean from the preliminary TREC 2005 results. Unlike most other filters, CRM114 is not a fixed-purpose antispam filter; rather, it's a general purpose language meant to expedite the creation of text filters. The pluggable CRM114 architecture allows rapid prototyping and easy support of multiple classifier engines; rather than testing different cutoff parameters, the CRM114 TREC test set tested different classifier algorithms and learning protocols.

Introduction and History: The CRM114 Discriminator was initially begun as a rough design in 1998 as a hypothetical filter design, meant to categorize incoming mail into a significant number of different "topics" (and with spam as an acknowledged target, but by no means the sole target). One of the design assumptions that CRM114 has held from day 0 has been that single word features are not as important as N-feature tuples. The initial CRM114 test code worked on N-letter tuples, rather than N-word tuples, but the groundwork had been laid. Testing against the very poorly camouflaged spam of 1999, the letter-tuple code showed an accuracy in excess of 98%, providing convincing evidence that the concepts of tuple features being substantially superior to single-token features and that a "no preconceived notions" machine-learning classifier could converge to superior accuracy over any human-designed heuristic system.

Work on CRM114 progressed past the initial EMACS macros and shell scripts that proved the concept to a C-coded multi-class classifier in 2000. With this came the realization that most of the work of filtering and categorizing email was not in the classifier itself- it was in massaging the inputs and outputs to conform to the vary large number of individual configurations, mail transfer agents, mail delivery agents, mail user agents, and user desires. Rather than wrap the learning and classification functions inside voluminous shell scripts, an initial look at making a Perl module that performed the N-tuple, N-class classification was made. However, Perl being what it was, the idea of a standalone

language made for manipulating the long overlapping strings needed for text classification became much more tenable. The first general public release of CRM114 under the Gnu Public License (GPL) occurred in early 2001, although "friends and family" releases had occurred since the beginning of the project.

This initial release has been followed by "whenever it's time" releases ever since. Typically CRM114 major releases occur three to six times a year, with bug-fix and "secret new feature" releases being done roughly monthly (or whenever a significant advance occurs). The releases are coded with both a date of release, and a nickname; the nickname often indicates the major contributor or motivator for that release. New releases have included multiple classifier engines (the current release has *seven* different classifiers built in) as well as major improvements such as TRAP/FAULT support, a JIT compiler, a preprocessor, and full 8-bit-safe data paths.

Overview of the CRM114 language: The CRM114 language is described in depth in the manual CRM114 Revealed [Yerazunis 2005], available for free download from the CRM114 web page at http://crm114.sourceforge.net. The language is a command based language with a declensional (rather than positional) syntax, and only one data type – the overlapping string. All CRM114 functions are designed to operate efficiently on strings that may contain other partially overlapping strings; this includes efficient reclamation of unused string space. The language contains only overlapping strings; there are no specific constructs for numbers, pointers, continuations, coroutines, open files, or anything else except overlapping strings.

The CRM114 language contains specific constructs for string-oriented matching, alteration, iteration, decision, and translation, however in the current context the most important statements are the LEARN and CLASSIFY statements. LEARN and CLASSIFY accept a uniform set of parameters and properly map those parameters onto whichever classifier is requested. Thus, it's possible to change classifiers in a CRM114 program (or even compare different classifiers, such as a Bayesian and a Winnow, or a Markovian and a Hyperspatial) without having to change the rest of the support structure.

Most of the CRM114 classifiers start with the concept of a word tuple as the basic feature to be considered. The tuple is defined as a series of words, and a word is defined as whatever matches a programmer-supplied regex (the default is that a word is one or more printable characters surrounded by whitespace). CRM114 kits include the very efficient TRE approximate regex engine [Laurikari 2001][Laurikari 2004] allowing a large amount of freedom in defining a word.

Structure of a CRM114 classifier: CRM114 classifiers are designed to be "pluggable"; a new classifier can be written in as little as a few hours by using a basic classifier code as a skeleton. A classifier is given a pointer to the unknown text, the unknown text's length, and a pointer to an argument parameter block containing any flags or word-definition regexes supplied; the classifier returns a condition code and optionally a textual representation of the results of the classification.

Almost all of the CRM114 classifiers use tuple-based features. That is, as words are tokenized by the regex engine, they are recombined into a series of short phrases of between one and five words long. The recombination sequence varies with the classifier and with the classifier flags; in newer versions of CRM114 any classifier can be configured to use single words as features with the `<unigram>` keyword; in the case of classifiers using Bayes rule this results in a Graham-style Bayesian filter (albeit one with an unlimited "peak window").

A more detailed description of the CRM114 classifiers may be found in [Chhabra 2004] and [Siefkes 2004]; we will give only a basic description here.

The original CRM114 classifier was the Markovian classifier; this classifier uses a Sparse Binary Polynomial Hashing (SBPH) feature generator that uses all in-order sequences of up to five words to form feature phrases. For example, the sentence "TREC is sponsored by NIST" will generate the following feature phrases:

TREC
TREC is
TREC <skip> sponsored
TREC is sponsored
TREC <skip> <skip> by
TREC is <skip> by
TREC is sponsored by
TREC <skip> <skip> <skip> NIST
TREC is <skip> <skip> NIST
TREC <skip> sponsored <skip> NIST
TREC is sponsored <skip> NIST
TREC <skip> <skip> by NIST
TREC is <skip> by NIST
TREC is sponsored by NIST

All sixteen of these phrases are then used as features; in the Markovian filter they are weighted unequally; longer phrases are given more weight than short ones [Yerazunis 2004].

The OSB, OSBF, Winnow, and Hyperspace classifiers all use a simpler feature generator which (paradoxically) simultaneously gives higher performance and higher accuracy. This feature generator is called OSB, for Orthogonal Sparse Bigram; it generates only four features instead of sixteen. Using the same example text "TREC is sponsored by NIST", the features are:

TREC is

TREC <skip> sponsored
TREC <skip> <skip> by
TREC <skip> <skip> <skip> NIST

Note that the unigram feature (single words taken one at a time, in isolation) is not used in the OSB feature set; extensive testing shows that presence of the unigram does not improve results; in fact, it sometimes causes a paradoxical decrease in accuracy.

After generating the features, a lookup is done into the stored statistics files. In order to accelerate this lookup, CRM114 does not use a general-purpose database such as MySQL or PostgreSQL. Instead, the feature is converted to a 64-bit hash, and that hash value used as the starting index for searching an in-memory hash table index. On the average, this lookup takes only six adjacent 32-bit reads from memory (assuming that the feature hasn't been seen before and isn't in the CPU L1 cache). Almost all CRM114 classifiers use this hashing scheme to accelerate feature lookups. As a secondary effect, the actual text is not stored; this gives a modicum of security to the statistics files (although individual features would fall quickly to a dictionary attack, the sequences would be more difficult, and of course, it would be impossible to recover any particular learned text with certainty.).

Once the individual feature lookups are performed, the counts are multiplied by the appropriate feature weight and an appropriate set of conversion and combining rules are used to convert individual feature frequencies into a final judgment. In Markovian, OSB, and Bayesian operation which by default use a Bayes Rule-like combining law, the conversion from frequency-of-occurrence to a priori probability is:

$$P_{local} = 0.5 + \frac{\text{in\_class occurrences} - \text{out\_of\_class occurrences}}{16 * (\text{total occurrences} + 1)}$$

Note that this produces a local probability that is strongly biased toward 0.5 (that is, toward uncertainty). These probabilities are weighted and combined via this modified Bayes chain rule to yield the output probabilities:

$$P_{final\_class\_i} = \frac{P_{initial\_class\_i} * P_{local\_class\_i}}{\text{SUM}_{\text{over all classes}} (P_{initial\_class\_j} * P_{local\_class\_j})}$$

These probabilities are then normalized. Additionally, two manipulations are performed – to avoid floating-point underflow, the value of the largest probability is renormalized at each step from the sum of the smaller probabilities, and for the purposes of human-readable output, the result is converted to pR; pR is the base-10 log of the probability ratio, and given IEEE floating-point inputs, varies from -340 to +340. This allows expression of a very large range of numbers in human readable terms.

CRM114 also has some support for a chi-squared decision rule, however that support was not used in the current NIST TREC test set.

Filter Configurations As Tested: The four filter configurations for CRM114 were OSBF, Winnow, OSB Unique, and "plain" OSB. Because CRM114 is a language, not just a filter, changing between these four variants only requires setting a parameter variable to the CLASSIFY and LEARN statements. The version of CRM114 used was CRM114-20050518.BlameMercury, (except for the Winnow version, which uses the Java implementation created by Christian Siefkes, available at http://www.inf.fu-berlin.de/inst/ag-db/software/tie/ because of a suspected minor bug in the CRM114 implementation that causes very slight discrepancies between what should be identical programs).

Additionally, all four variants enabled some form of "microgrooming", a method of aging out old data from the statistics files to keep the statistics files from growing without bound. Microgrooming is based on the concept that when it is necessary to make room in the database file, and given two seldom-seen features, one old and one new, get rid of the older one first as it has a longer history of not being useful. Because CRM114's hash storage structure uses linear in-table overflow chaining, this information is indirectly available as the distance each feature is actually found in, compared to its nominal location (i.e. the further down the overflow chain that we actually find a feature compared to where the feature would have been placed if no overflow was present, the newer the feature is.). The Java Winnow system used exact explicit rather than implicit approximate aging; results are similar.

All four filter configurations use a phrase-creation window of length 5 and OSB-style features. By using these N-word tuples, OSB features provide a Markov Random Field model rather than a Bayesian model of the texts learned [Chhabra 2004].

None of the CRM114 filters use any preprocessing whatsoever. There is **no** stop word removal, **no** tagging of items in the headers versus those in the body, and **no** decoding of any type, not even MIME decoding. All four classifiers operated on the "as seen" texts, without any further interpretation, and the default word defining regex was used (specifically, [[:graph:]]+ defines a word except in the Java Winnow code which used the X tokenizer from [Siefkes 2004]; the X tokenizer is XML/HTML-aware and is expressed as [^\p{Z}\p{C}][/!?#]?[-\p{L}\p{M}\p{N}]*(?:["'’=;]|/?>|:/*)? using Unicode categories.

Descriptions of individual filters: We will now describe the four configurations tested.

OSBF: OSBF is a variant of OSB that uses confidence factors that vary depending on the number of documents learned, The confidence factor is used to influence the local per-feature probability and distance-weighting of the features is used, with an inverse-exponential weighting as distance increases. Beyond that, OSBF uses Bayes rule to combine the local feature probability (a gross abuse of Bayes' assumptions of independence that most filter authors simply live with). Unlike most Bayesian spam

filters, there is no "top 10" window of the most extreme values; all values are used, no matter how insignificant. Unfortunately, the version of OSBF used in these tests contained a significant bug in the chain rule, hence the values reported below are much worse than the algorithm is capable of. The training was configured so that if a text did not score at least +10 pR units in the correct class, then the text was trained, whether or not it actually was classified correct by some narrow margin. The first 500,000 characters of each text were used.

WINNOW: WINNOW is an implementation of Nick Littlestone's Winnow algorithm. Winnow is comparable to a back-propagating perceptron- individual features start out with a weight of 1.0000 and are multiplicatively promoted or demoted according to whether the feature is found or not found in a particular text being learned. Instead of Bayes rule, WINNOW uses a simple summing of the weights; the class with the highest weight count wins. Our Winnow implementation uses a thick threshold for learning. Documents are trained not only in case of mistakes, but also if the determined score was near the threshold. This makes the classifier more robust when classifying borderline instances. See [Siefkes 2004] for a detailed description.

OSB and OSB Unique: OSB and OSB Unique use the basic local probability formula above without confidence factors; like OSBF they use all local probability values (there is no extremum window). Feature counts are weighted in an empirically derived decreasing sequence. There is only one difference between OSB and OSB Unique: whether a repeated feature is allowed to further influence the output or is simply discarded. Among all tested classifies, only OSB uses all repeated features. OSB Unique discards repeated features, as do OSBF and Winnow. In both OSB and OSB Unique, a single-sided thick threshold for training of +20 pR units was used; any text not scoring at least +20 pR units in the correct class was trained. Only first 4096 characters of each text were used for classification; this is the default for one of the more commonly used CRM114 mail filters deployed.

Analysis of the TREC pre-conference partial release results: The following four tables show how CRM114 fared with respect to the pre-conference data release. 1-ROCAC% and Average LAM% are only prepublished as median values; final LAM% is not prepublished at all (note that unlike some filters, none of the CRM114 configurations contained <u>any</u> preloaded learning. Everything was learned strictly on the fly.) Despite this "no preload", all of CRM114's classifiers beat both the pre-publication median 1-ROCAC% and LAM%, often by an order of magnitude.

The design of most CRM114 classifiers assumes a "balanced error" configuration- that all misclassifications are equally bad. Although common wisdom is that a false reject is more dangerous than a false accept, at least one author considers this incorrect. Consider the situation of a well-done phishing spam being falsely accepted. Then, consider it being believed by a nontechnical computer (say, a grandmother); the likely resulting financial costs of this one false accept will approach if not exceed the likely financial costs of a false reject of a typical legitimate financial notification. Thus, to a first approximation, we consider that all misclassifications are roughly equally costly and an equal-

error-rate configuration is a good loss minimization guideline.

To examine the behavior of filters operated in this equal-error configuration, we consider two "sweet spots" that our experience show are reasonably representative of equal-error operation- specifically, what the error rates are for spam when the good error rate is fixed at 1%, and what the error rate for good is when the spam error rate is fixed at 1%. These representative "sweet spots" are shown as the last two columns in the charts. The "sweet spot" column header shows three values – the best value for any filter tested, the median value, and the worst value. A sweet spot entry in **BOLD TYPE** is used to highlight where CRM114 performance was either "best in class" or "statistically indistinguishable at the 95% confidence level from best in class" over the entire set of 44 filter configurations tested.

As shown below, for every corpus, and for every "sweet spot", at least one of the four CRM114 configurations was either "best", or statistically indistinguishable from "best". Interestingly, no one configuration of CRM114 was consistently best. This may well be a manifestation of the No Free Lunch theorem in decision and optimization [Wolpert 1997]

MRX Corpus: The MRX (Mr. X) corpus is a "single person" corpus sponsored by Prof. Gordon Cormack of Waterloo University. It is biased toward spam – it contains about 9K good emails and 40K spams: (about 18% good emails). Here we see that Winnow, operating with the OSB feature set, was statistically indistinguishable in performance with the best of the 44 filter configurations submitted to TREC. It also had an impressively small 1-ROCAC% of just 0.051.

MRX corpus	***1-ROCAC% median= 1.613***	***AVG LAM% median = 2.53***	***Final LAM%***	***Ham 1.0% 0.34 24.24 99.65***	***Spam 1.0% 0.37 13.35 99.16***
OSBF	0.311 (0.244 - 0.397)	1.46 (1.34 - 1.59)	1.4 (1.13 - 1.62)	3.85 (2.82 - 5.22)	2.85 (2.49 - 3.27)
OSB Winnow	0.051 (0.035 - 0.075)	0.60 (0.53 - 0.68)	0.24 (0.17 - 0.33)	**0.43 (0.31 - 0.62)**	**0.51 (0.37 - 0.69)**
OSB Unique	0.177 (0.128 - 0.246)	0.98 (0.85 - 1.13)	0.23 (0.12 - 0.42)	1.07 (0.86 - 1.35)	1.07 (0.83 - 1.39)
OSB	0.218 (0.157 - 0.304)	0.79 (0.69 - 0.90)	0.34 (0.23 - 0.52)	0.63 (0.50 - 0.79)	0.56 (0.42 - 0.75)

FULL corpus: The FULL corpus is a fairly balanced corpus. It contains 39K good emails and 53K spams (that's 42% good email). In this corpus, the plain OSB (that is, a tuple-based Bayesian filter, using no prefiltering or other "gimmicks", was the best filter tested across all TREC submissions, for both sweet spots. Interestingly, omitting replicated features degraded filter performance to barely statistically significant levels (the error bars still overlap, but the central values are not covered in the

opposite configuration's error bars.). Note that in this corpus, OSB beats OSB Unique in everything but 1-ROCAC% (0.049 versus 0.042) – keep this in mind as you read the next corpus.

FULL corpus	***1-ROCAC% median= 0.861***	***LAM% median = 2.02***	***Final LAM%***	***Ham 1.0% 0.23 10.35 94.44***	***Spam 1.0% 0.15 14.20 98.11***
OSBF	0.169 (0.151 - 0.190)	1.74 (1.66 - 1.83)	3.6 (3.28 - 3.97)	3.14 (2.81 - 3.50)	3.15 (2.94 - 3.38)
OSB Winnow	0.122 (0.102 - 0.145)	0.73 (0.68 - 0.79)	0.86 (0.72 - 1.02)	0.68 (0.59 - 0.79)	0.51 (0.42 - 0.62)
OSB Unique	0.042 (0.031 - 0.056)	0.63 (0.56 - 0.70)	0.43 (0.33 - 0.56)	0.35 (0.30 - 0.42)	0.21 (0.16 - 0.27)
OSB	0.049 (0.035 - 0.068)	0.47 (0.43 - 0.52)	0.37 (0.29 - 0.47)	**0.23 (0.19 - 0.27)**	**0.15 (0.11 - 0.20)**

SB Corpus: The SB corpus is a smaller corpus with 6K good emails and only 775 spams (88% good); thus it is heavily weighted toward filters with a prejudice to accept borderline emails as good. Both OSB and OSB Unique did very well in this corpus – both are statistically indistinguishable from the best of the 44 filters submitted to TREC. However, OSB Unique seems to have a slight edge in 1-ROCAC%, LAM%, and Final LAM%. Note that this is an inversion from the FULL corpus, where OSB was better than OSB Unique.

SB corpus	***1-ROCAC% median= 2.845***	***LAM% median = 4.79***	***Final LAM%***	***Ham 1.0% 3.48 27.23 100.00***	***Spam 1.0% 3.79 63.76 99.52***
OSBF	2.393 (1.689 - 3.382)	2.34 (1.87 - 2.93)	0.91 (0.56 - 1.47)	8.77 (7.05 - 10.88)	97.11 (57.03 - 99.88)
OSB Winnow	1.888 (1.315 - 2.704)	2.14 (1.68 - 2.73)	1.2 (0.73 - 1.88)	10.32 (8.32 - 12.74)	62.35 (37.84 - 81.84)
OSB Unique	0.231 (0.142 - 0.377)	1.64 (1.28 - 2.10)	0.59 (0.34 - 1.03)	**3.48 (2.26 - 5.32)**	**4.01 (2.40 - 6.63)**
OSB	0.393 (0.203 - 0.759)	1.67 (1.32 - 2.12)	0.44 (0.24 - 0.80)	**3.74 (2.32 - 5.97)**	**5.62 (1.99 - 14.86)**

TM Corpus: The TM corpus is also highly biased toward nonspam emails- it contains 150K good

emails and 19K spams (88% good emails). Here OSB is statistically clearly better than OSB Unique; the error bars don't even overlap. What's also of interest is that Winnow beats OSB in 1-ROCAC%, LAM%, and ties it in Final LAM%, yet the 95% confidence interval error bars for OSB and Winnow barely touch, with OSB being "better" in both sweet spots (apparently equal to the best of all 44 filters submitted to TREC.)

TM corpus	***1-ROCAC% median= 2.712***	***LAM% median = 2.54***	***Final LAM%***	***Ham 1.0% 1.04 10.28 99.06***	***Spam 1.0% 1.07 47.70 99.40***
OSBF	0.790 (0.720 - 0.868)	2.44 (2.33 - 2.56)	2.5 (2.28 - 2.80)	10.28 (9.71 - 10.89)	16.93 (13.00 - 21.74)
OSB Winnow	0.166 (0.138 - 0.201)	0.79 (0.74 - 0.85)	0.46 (0.39 - 0.54)	1.30 (1.15 - 1.46)	1.86 (1.34 - 2.57)
OSB Unique	0.195 (0.160 - 0.238)	1.08 (1.01 - 1.14)	0.68 (0.59 - 0.78)	1.58 (1.40 - 1.78)	1.71 (1.49 - 1.97)
OSB	0.272 (0.225 - 0.329)	0.83 (0.77 - 0.89)	0.46 (0.39 - 0.54)	**1.04 (0.92 - 1.16)**	**1.07 (0.86 - 1.34)**

Conclusions: As can be seen, one or more of CRM114's classifiers was either the best or statistically indistinguishable from the best in all eight sweet spots (4 corpora x 2 sweet spots/corpus). However, interesting inversions occurred in things like 1-ROCAC% and rate of learning.

It seems that the No Free Lunch theorem [Wolpert 1997] is alive and well with a vengeance in the spam filtering world; statistically significant variation exists with even simple changes like whether to allow repeated features as classifier input or not.

The design decision to make CRM114 a "learning" system versus a "prelearned" system seems to be justified; at least one CRM114 classifier learned fast enough to win every 1% sweet spot in every corpus.

The design decision to use a hash-based statistics system rather than a database seems to have been justified; because the hash-based system is so fast, CRM114 developers can feasibly test hundreds of variations when other developers can only test two or three different ideas.

The design decision to make CRM114 a language rather than a single-purpose spamfilter tool seems to been justified; because the language allows easy creation of multiple coexisting variants that can be tested against each other easily.

References:

[Chhabra 2004] Shalendra Chhabra, William S. Yerazunis, and Christian Siefkes. "Spam Filtering using a Markov Random Field Model with Variable Weighting Schemas", in Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM '04), 2004.

[Laurikari 2000] Laurikari, Ville, "NFAs with Tagged Transitions, their Conversion to Deterministic Automata and Application to Regular Expressions", Symposium on *String Processing and Information Retrieval* (SPIRE 2000), September 2000. (downloadable from http://laurikari.net/ville/cv.html)

[Laurikari 2004] Laurikari, Ville, TRE approximate regex library – free download at http://laurikari.net/tre/index.html;

[Siefkes 2004] Christian Siefkes, Fidelis Assis, Shalendra Chhabra, and William S. Yerazunis "Combining Winnow and Orthogonal Sparse Bigrams for Incremental Spam Filtering", *European Conference on Machine Learning (ECML) / European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, September 2004. Download at: http://www.cs.ucr.edu/~schhabra/winnow-spam.pdf

[Wolpert 1997] David H. Wolpert and William G. Macready, "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation, Vol 1, number 1, pp 67-82, April 1997

[Yerazunis 2004] William Yerazunis, "The Spam-Filtering Accuracy Plateau at 99.9% Accuracy and How to Get Past It", The MIT Spam Conference 2004, available for download at: http://crm114.sourceforge.net/Plateau_Paper.pdf

[Yerazunis 2005] William Yerazunis, CRM114 Revealed – Or How I learned To Stop Worrying and Trust My Automatic Monitoring Systems , this is the complete CRM114 manual. Available for free download at http://crm114.sourceforge.net