SPAM Detection: Naïve Bayesian Classification and RPN Expression-based LGP Approaches Compared

Clyde Meli[1] and Zuzana Kominkova Oplatkova[2]

[1]CIS Department, Faculty of ICT, University of Malta, Malta
Clyde.meli@um.edu.mt

[2]Department of Informatics and Artificial Intelligence, Faculty of Applied Informatics
Tomas Bata University in Zlin
Nam. T.G.Masaryka 5555, Zlin, Czech Republic
oplatkova@fai.utb.cz

Abstract. An investigation is performed of a machine learning algorithm and the Bayesian classifier in the spam-filtering context. The paper shows the advantage of the use of Reverse Polish Notation (RPN) expressions with feature extraction compared to the traditional Naïve Bayesian classifier used for spam detection assuming the same features. The performance of the two is investigated using a public corpus and a recent private spam collection, concluding that the system based on RPN LGP (Linear Genetic Programming) gave better results compared to two popularly used open source Bayesian spam filters.

Keywords: Reverse Polish Notation (RPN), naïve Bayesian classifier, spam detection, Linear Genetic Programming (LGP), Genetic Programming (GP)

1 Introduction

1.1 Overview

The paper deals with the spam detection based on the Reverse Polish Notation (RPN) [1], [2] expression based Linear Genetic Programming (LGP) [3]–[5] approach compared to Naïve Bayesian classifier [6]–[8].

Traditionally spam was fought using blacklists and heuristics [9]. In recent years, the spam war has had the help of machine learning and text classification methods. The war has not been won yet. In 2012, a record number of 27 million new malware were found, according to Panda Security [10]. A study [11] reported in 1998 that 10% of incoming messages on a LAN were spams. Spam is an economic expense and it slows down legitimate traffic.

One main frontline against spam has recently been the statistical approach using the Naive Bayesian classifier [6]–[8]. Paul Graham in "A Plan for Spam" [12] and "Better Bayesian Filtering" [13] presented his algorithm and implementation of the Naïve Bayesian classifier, the latter with a more complicated tokenizer. Academic implementations include Pantel & Lin's SpamCop [14] and Sahami et al's Bayesian

adfa, p. 1, 2011.

filter [15]. SpamAssassin [16] is a well known open source implementation of this algorithm. Günter Bayler [17] discusses a number of attacks on Bayesian spam filters in his book, which exploit their weakness.

For the Bayesian classifier to be effective, it requires a large number of records. Secondly where a predictor category is not present in the training data, Naïve Bayes takes the assumption that a new record with that category would have 0 probability. If such a predictor category is important, this can be a problem. As a result when the goal is to determine class membership probability, Bayes is very biased, and as a result rarely used in Credit Scoring [18].

1.2 Related work

Different techniques were developed for the fight with spam detection from various field of machine learning.

Sangeetha et al. [19] proposed a local concentration (LC) based feature extraction approach for spam detection. Goweder et al. [20] developed a system to detect spam based upon a neural network, used as a classifier, whose weights were evolved by a GA. Artificial immune systems such as described by Khorsi [21] have been used to fight spam and computer viruses by imitating biological immune systems.

Katirai [22] proposed a method for junk email classification which used the classical form of Genetic Programming (GP) to automatically evolve a Bayesian filter. The Bayesian filter program was represented by a syntactic tree whose nodes are numbers representing word frequencies, operations on numbers, words and operations on words. The program would evolve the better filter program according to a fitness function developed to minimise misclassifications, ie. the sum of squared errors over all documents.

Hirsch et al. [23] implemented a GP system to evolve rules based on n-grams (strings n character long) for document classification. GP has been used by Shengen et al. [24] to generate new features used as inputs to Support Vector Machines (SVM) and GP classifiers to detect link webspam.

Earlier systems include Magi [25] which used decision trees to automatically route new messages to the relevant folders, RIPPER [1] which automatically learnt rules to classify email into categories (spam was not mentioned), and Genetic Document Classifier [2] which using a classical GP routed inbound documents to interested research groups within a large organisation.

Davenport et al. [26] implemented a GP system which evolved a Reverse Polish Notation postfix representation rather than the classical trees. The operators included logical operators.

Various spam corpora exist such as SpamBase [27], SpamAssassin and Untroubled.Org. SpamBase was not used in this study because it required the use of specific features, the raw emails were not available and use of more fresh emails was preferred.

2 Methods

2.1 Bayesian Approach

The Bayesian Network is defined as a directed acyclic graph compactly representing a probability distribution [28]. Nodes represent a random variable F_i. A directed edge between two such nodes indicates a dependency or probabilistic influence. It is assumed by the network structure that each node Fi in the network is conditionally independent of its non-descendants. Thus each node Fi in the network is associated with a conditional probability table which specifies the distribution over Fi.

A Bayesian classifier is a Bayesian network applied to classification, containing a node C representing the class variable and a node Fi for every feature.

From Bayes Theorem, given feature variables f_1, f_2, .. f_n, the Bayesian network is used to compute the probability that a document k with vector F=<f_1, f_2, .. f_n> belongs to category c is (1):

$$P(C = c|F = f) = \frac{P(C = c)P(Fi = fi|C = c)}{\sum_{k\in\{spam,ham\}} P(C = k)P(Fi = fi|C = k)} \tag{1}$$

The Naïve Bayesian Classifier assumes that $f_1, f_2, .. f_n$ are conditionally independent, which gives (2):

$$P(C = c|F = f) = \frac{P(C = c)\prod_{i=1}^{n} P(Fi = fi|C = c)}{\sum_{k\in\{spam,ham\}} P(C = k)\prod_{i=1}^{n} P(Fi = fi|C = k)} \tag{2}$$

where both P(Fi|C) and P(C) are relative frequencies which can be calculated from the training corpus.

It is possible to define a criterion in (3) and further in (4) by development of the idea that mistaking a ham message (legitimate) as being spam is much more severe as a mistake than the opposite (mistaking spam as ham). If it is assumed that misclassification of ham as spam is *L* more times costly than misclassification of spam as ham, that the independence criterion holds and that the probabilities have been estimated correctly. Thus a message can be classified as being a spam if it complies with (3).

$$\frac{P(C = spam|F = f)}{P(C = ham|F = f} > L \tag{3}$$

Following Sahami et al's formulation [15] using P(C=spam|F=f)=1-P(C=ham|F=f), the criterion can be rewritten as (4):

$$P(C = spam|F = f) > t, where\ t = \frac{L}{1 + L}, L = \frac{t}{1 - t} \tag{4}$$

Sahami et al. [15] set the threshold *t* to 0.999 and *L* to 999, meaning that blocking a ham message as spam is as bad as letting 999 spam messages through the filter. There are cases as indicated by Sahami et al. where it would be feasible to use lower thresh-

old, such as in the case of having a different action instead of blocking detected spams; e.g. sending a challenge to the sender in the case of a potential spam. Sahami has noted the independence assumptions are often violated in practice, resulting in poor performance. As a threshold, Graham [13] uses 0.9 with the argument that few probabilities end up in the midrange.

2.2 Genetic Programming

Genetic Programming (GP) [29]- [31] is a technique used to solve problems by methods based upon evolutionary mechanisms inspired from biology applied to the evolution of computer programs. Using Genetic Algorithms (GA), every individual represents a program and the fitness function depends on how good it solves the task represented by the fitness function.

Linear Genetic Programming systems (LGP) [3]–[5] which are extensions of the classical GP take a different approach. The program which is evolved is represented using a sequence of instructions in either machine language or imperative programming language. This makes it easier to use classical genetic operators like bit mutation and crossover as opposed to the classical GP as well as avoiding the standard GP's interpretation stage in the case of a machine opcode implementation. It also has been found superior over a set of benchmark problems (to the classic GP) [3], [32], [33]. A typical LGP program would have registers and constants from pre-defined sets.

Various applications of LGP's have been implemented, including web usage mining [34], formulation of the compressive strength of concrete cylinders [35], time-series modelling [36], and intrusion detection [37], [38].

2.3 RPN Expression

For the purpose of comparing Naïve Bayesian classification and Reverse Polish Notation Expressions, where the latter are made up of operators (such as +, -, *, /, sine, cosine, AND, OR and XOR) and operands (constants and evaluated feature values Fi), we assume that Fi are the same for the RPN Expression as for the Naïve Bayesian classifier.

It is known that the Naïve Bayesian classifier is incapable of expressing propositional operators such as XOR [39]. However, the attributes or features are independent, they act independently to classify and they do not interact. Thus, they cannot capture concepts like XOR (similar to the limitation of single-layer perceptrons).

On the other hand, a Reverse Polish Notation [40], [41] expression can easily be used to express propositional logic. Reverse Polish Notation was proposed by Burks, Warren and Wright [42], based on Polish Notation which was invented by logician Jan Lukasiewicz in the 1920's. One can express propositional logic with an RPN expression by using specific AND, OR and XOR operators or by using integer arithmetic operators (e.g. XOR can be built from modulus, sum and multiply operators, and modulus itself can be built from division, multiplication and subtraction operators).

Galculator [43] is an example implementation of an algebraic and RPN calculator with propositional operators.

One advantage of Reverse Polish Notation is that there is no need to use parentheses which are sometimes needed. 3-x+y would be written as 3 x – y + using RPN. 4+x*y also can be written as 4+(x*y). However changing the location of the parenthesis would result in something very different, i.e. (4+x)*y. The parentheses are used to force order. On the other hand, the RPN postfix equivalent of the former is 4 x y * +. Calculations with RPN are performed faster using an iterative interpreter than the equivalent infix.

Evidence has been found [44] that Naïve Bayes produces poor probability estimates (Bennett writes that it tends to produce uncalibrated probability estimates). Monti and Cooper [45] similarly provided evidence that the Naïve Bayes model is poorly calibrated.

Thus based on this, it is to be expected that Reverse Polish Notation Expressions are more expressive than Naïve Bayesian classification. Implementation of a system based on RPN expressions should be as good as one based on Naïve Bayes.

2.4 Implementation overview

Based on the preceding, it was expected that RPN Expressions might give better results in a practical implementation than the use of an equivalent Naïve Bayesian classifier. The idea was to build an LGP [4] system which evolves a program in the form of an RPN Expression (using machine code format), which would be able to be used as a detector of spam in emails. Once a sufficiently good RPN expression would be evolved, it would be able to be used for classification apart from the LGP system.

The system was implemented by the author. It utilises the feature extraction approach using RPN's and uses multi-threading. It was assumed that alphanumeric characters (including non-English characters), dashes, apostrophes, euro and dollar symbols are to be considered part of tokens, and any left-over symbol is taken to be a token separator. No stopword filtering was performed.

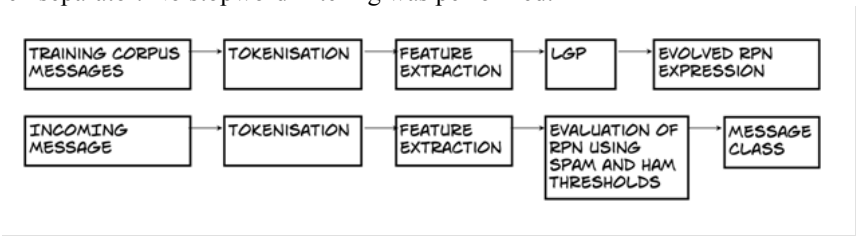

Fig. 1. Training and Classification Phases of the RPN / LGP model

Figure 1 shows the training and classification phases of the RPN/LGP model. After the tokens are recognised, a number of features are extracted and then used within an RPN expression made up of feature values as represented by the current LGP chromosome. A stack is used to evaluate the RPN expression.

The features used are grouped in four. 1) A group of features are evaluated on the subject line; 2) a group on the Priority and Content-Type headers; 3) a group on the whole message body; 4) a group of features are evaluated on any URL found within the body. The latter features include an evaluation of Internet browser services WOT [46] and usage of the Google Safe Browsing API [47] as well as DNS RBL [48] checks. Some of the other features included Yule's Measure [49], Hapax Legomena and Simpson's Measure [50], plus some new measures developed and implemented by the author inspired from text measures used with author attribution. Characteristics of texts have been examined for obtaining their mathematical characteristics and for document clustering [51], [52]. One measure evaluates Zipf's Law [51] and the other a metric similar to Zipf's Law. Extra mechanisms like transcription/repair of chromosomes inside the optimization algorithm were not needed. When evaluating an expression, attempts to pop an empty stack return UNDEF (undefined), meaning that operators requiring two operands would evaluate to UNDEF. Selection pressure removed such expressions from the population. More details on features was not included due to space constraints in this paper.

The implementation was coded under Windows and Linux operating systems in C++ using the Boost library for the bit string implementation. The LGP's GA was initialised using the Mersenne Twister [53]. The fitness value is stored to speed up evaluation of fitness values for an unchanged chromosome. Changes to a chromosome will trigger reevaluation. Under Linux a DNS caching server [54] was used to speed up the system.

In the LGP implementation, single point crossover is performed with Elitism, so the best chromosome always survives. The implemented selection operator uses Holland's Proportion Fitness Selection. "Per locus" (per bit) mutation is specified rather than "per chromosome", as in De Jong & Spears [55]. Linear chromosomes rather than trees were implemented using binary opcodes.

Fitness determines how good an RPN expression (chromosome) detects spam and ham correctly using this algorithm (5) and (6):

Incorrectly classed = number of incorrectly recognised ham from ham corpus + number of incorrectly recognised spam from spam corpus. (5)

$$\text{Fitness value} = \text{MAXSPAMHAM} - \text{Incorrectly Classed} \quad (6)$$

MAXSPAMHAM is a constant which is equal to the total number of spams and hams in the corpus. The objective was to maximize fitness.

For every email, multiple features are evaluated and the values are cached avoiding unnecessary computation. The evolved RPN expression is evaluated with the results of the features and other constants. Two static thresholds are utilised with the result from the RPN expression for determining whether the email is a spam or a ham, in a similar way to Zhang and Cieselksi's GP object classification system [56]. It is also reminiscent of Paul Graham's two probability thresholds for spam and ham [13]. The thresholds were chosen arbitrarily initially and justified by tests, which included testing of different threshold algorithms.

Finally, the final or best thresholds used were 0 for spam and 1000 for ham. Test results here used the 40 value for ham which was as good as using the 1000 value.

The following was the final threshold algorithm used:

if we KNOW this is supposed to be a spam (supervised learning),
if evaluation<SPAM THRESHOLD then increment wronglyclassed;
else (in the case of ham) if (evaluation<HAM THRESHOLD && evaluation>SPAM THRESHOLD) then increment wronglyclassed

LRU (Least Recently Used) Caching was used to speed up the system, and caches can be saved and reloaded. The caches should be cleared from time to time because feature values based on Internet services like dns or WOT will change in time (a legitimate domain may expire and become a malicious website).

The Labelled Training Set used during various testing phases was the following:

a) 280 Personally collected spam and ham, from personal mailboxes and mailing lists

b) 610 Spams from the public untroubled.org spam corpus (which only contains spam)

An RPN expression is made up of machine code opcodes, each representing an operand or an operator. Operators used were +,-,*,/,sine and cosine. Registers are Feature detector values evaluated on the email. Operands are registers or constant values.

The LGP is run with a training collection of emails already classified as spam or ham. This did not include attachments. Emails were stored in separate folders for spam and ham.

RPN Example 1: f[1] f[2] f[3] f[4] f[5] f[6] * * * * *
RPN Example 2: f[1] f[2] * f[3] + 10 * 5 –

Fig. 2. Examples of RPN Expressions used in GAGENES/LGP, where f[n] is the value of the feature function no n, evaluated on the current email being tested

The opcode bits determine whether it represents an operator, register or constant. If the top three bits are equal to '000', then the rest of the opcode specifies a register number, if it is '001' then the rest is a constant value, otherwise it is taken to be an operator (from 010 to 111).

3 Results

This study was looking for an effective method for email spam detection. The feature extraction approach with evolved RPN expressions was used.

During simulations, the following settings was applied: Population size in LGP was set to 15, Mutate probability to 0.5, Crossover probability to 0.8 and Copy probability to 0.001. Based on results, it can be stated that this settings - high mutation rates like 0.5 and a small population – performed good solutions. It is corroborating what was found in the GA literature [57] about high mutation rates.

Every test was repeated 5 times with the same parameters. Test platform ran Windows 7 and Gentoo Linux, with an Athlon 64-bit Phenom II X6 2.86Ghz. The number of fitness evaluations per run was equal to 5*15; evaluation resulted in working out the feature values for every email which were then cached, speeding things up. Originally, the average population fitness tended to decrease throughout generations, however it tended to remain stable. A good chromosome tended to dominate early in the population. Thus the Selection algorithm was modified to select only non-zero fitness chromosomes, which helped increase population average fitness, though some bloat was exhibited in at least one test.

A test corpus of 3657 emails which included 280 hams was used to test the evolved RPN expression. The test corpus was built as follows:

1) 280 Hand-chosen Hams taken (from the Training Corpus)
2) Another 32 Hand-chosen Hams
3) 607 Hand-chosen Spams
4) 2763 Spams from the Untroubled.org archive (dated 2012)

LGP was compared with two popularly used and capable open source spam filters, SpamAssassin and BogoFilter. It is to be noted that SpamAssassin includes a Bayes filter derived from Graham and Robinson's proposals (it is a hybrid of Bayes plus a Rule-based system), whereas BogoFilter is a pure bayesian filter [58].

The same training set was used with BogoFilter. The three filters used the same testing set.

BogoFilter was not able to work with small training sets as was noted above. SpamAssassin was used with its default configuration. All emails were stored with full headers and did not have attachments. A bash script was used to call filters, and keep count of spams and hams detected.

In 2004, Graham-Cumming [59] proposed an alternative way of counting spam filter accuracy by working out the following individual hit rates (7) and (8):

$$\textit{spam hit rate} = \textit{total number of correctly detected spams} / \textit{total number of spams received} \quad (7)$$

$$\textit{ham hit rate} = \textit{total number of incorrectly detected hams} / \textit{total number of spams received} \quad (8)$$

However the industry still works out spam filter accuracy by working out total number of correctly detected spams and hams divided by the total number of messages (9).

$$\textit{Accuracy} = (\textit{correctly detected spams} + \textit{correctly detected hams}) / \textit{number of messages} \quad (9)$$

Accuracy (using data from Table 1 for the number of emails, false positives, etc.) using (9) was found to be as follows and in graphically form in Fig. 3.:

SpamAssassin: 82% ((3682-663)/3682); BogoFilter: 26.67% ((3682-2700)/3682); RPN LGP: 100% ((3682-0)/3682).

Unsure spam emails are assumed to be hams, so the False Positives (FP) for Bogofilter amount to 568 (366+202 unsure) for TESTSPAM and 2132 (75+2057 unsure) for UNTROUB (This refers to spam taken from the untroubled.org corpus).

False Positives on ham were higher with SpamAssassin than with BogoFilter, whereas False Positives on spam were much higher with BogoFilter than with SpamAssassin. This is consistent with spam filter testing done by Claypool and O'Brien [60]. In the words of Paul Graham, "email is not just text; it has structure" [13], which is one reason why the LGP system uses different features per email structure. With the author's LGP system there were no misclassifications with the testing set. GAGENES is the author's base C++ genetic algorithm library.

Table 1. Summary of SpamAssassin (SA), Bogofilter (BF) and GAGENES/LGP (LGP) performance on spam and ham. FP stands for False Positives.

Mailbox	*# Emails*	*SA # FP*	*SA % FP*	*BF # FP*	*BF % FP*	*LGP # FP*	*LGP %FP*
HAM	*280*	*6*	*2.1%*	*0*	*0.0%*	*0*	*0%*
TESTHAM	*32*	*0*	*0.0%*	*0*	*0.0%*	*0*	*0%*
ALL HAM	*312*	*6*	*1.9%*	*0*	*0.0%*	*0*	*0%*
TESTSPAM	*607*	*481*	*79.2%*	*568*	*93.6%*	*0*	*0%*
UNTROUB	*2763*	*176*	*6.4%*	*2132*	*77.2%*	*0*	*0%*
ALL SPAM	*3370*	*657*	*19.5%*	*2700*	*80.1%*	*0*	*0%*

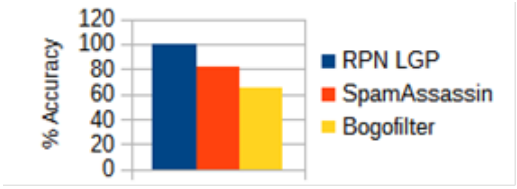

Fig. 3. Spam Filter Testing

4 Discussion and conclusion

This paper gives a brief introduction to the RPN expression-based LGP system of GAGENES/LGP applied to spam detection and the test results performed. It has been determined that the RPN representation utilised in this LGP system is a potentially good alternative to the classical Bayesian classifier. This study has found the feature extraction approach with evolved RPN expressions to be effective in email spam detection. The simulation proved that the proposed system has no misclassification within the used testing set of hams and spams.

To speed up fitness evaluation, feature results are cached and groups of features used multi-threading.

In future, the spam and ham thresholds may be evolved themselves. Also RPN expressions will be represented in a way which will eliminate invalid expressions. Future testing will involve k-fold testing and principal component analysis. The system may be extended to detect web spam, network intrusions and other malware.

Acknowledgement. Acknowledgements go to my Ph.D. Supervisors Dr Vitezlav Nezval. Thanks also to Tom Fawcett who answered my email query about the subject of Bayesian classifiers and RPN. This work was supported by Grant Agency of the Czech Republic - GACR P103/15/06700S, further by financial support of research project NPU I No. MSMT-7778/2014 by the Ministry of Education of the Czech Republic and also by the European Regional Development Fund under the Project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089.

5 References

1. Cohen, W.: Learning Rules that Classify E-Mail. In: In Papers from the AAAI Spring Symposium on Machine Learning in Information Access. pp. 18–25. AAAI Press.
2. Clack, C., Farringdon, J., Lidwell, P., Yu, T.: Autonomous document classification for business. In: Proceedings of the first international conference on Autonomous agents. pp. 201–208. ACM, New York, NY, USA (1997).
3. Brameier, M.: On linear genetic programming, https://eldorado.tu-dortmund.de/handle/2003/20098, (2004).
4. Brameier, M.F., Banzhaf, W.: Linear Genetic Programming. Springer (2006).
5. Brameier, M., Banzhaf, W.: A comparison of linear genetic programming and neural networks in medical data mining. IEEE Trans. Evol. Comput. 5, 17–26 (2001).
6. Androutsopoulos, I., Koutsias, J., Chandrinos, K.V., Paliouras, G., Spyropoulos, C.D.: An evaluation of Naive Bayesian anti-spam filtering. arXiv:cs/0006013. (2000).
7. Duda, R.O., Hart, P.E., Nilsson, N.J.: Subjective bayesian methods for rule-based inference systems. In: Proceedings of the June 7-10, 1976, national computer conference and exposition. pp. 1075–1082. ACM, New York, NY, USA (1976).
8. Mitchell, T.M.: Machine Learning. McGraw-Hill Science/Engineering/Math (1997).
9. Zdziarski, J.: Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification. No Starch Press (2005).
10. Reports | Press Panda Security, http://press.pandasecurity.com/press-room/reports/.
11. Cranor, L.F., LaMacchia, B.A.: Spam! Commun ACM. 41, 74–83 (1998).
12. Graham, Paul: A Plan for Spam, http://www.paulgraham.com/spam.html.
13. Graham, P.: Better Bayesian Filtering, http://www.paulgraham.com/better.html.
14. Pantel, P., Lin, D.: SpamCop: A Spam Classification & Organization Program. In: In Learning for Text Categorization: Papers from the 1998 Workshop. pp. 95–98 (1998).
15. Mehran Sahami, Susan Dumais, David Heckerman, Eric Horvitz: A Bayesian Approach to Filtering Junk E-Mail. Proc. AAAI-98 Workshop Learn. Text Categ. (1998).
16. SpamAssassin Homepage, http://spamassassin.apache.org/.
17. Bayler, G.: Penetrating Bayesian Spam Filters: Exploiting Redundancy in Natural Language to Disguise Spam Emails. Vdm Verlag Dr. Müller (2008).
18. Shmueli, G., Patel, N.R., Bruce, P.C.: Data Mining for Business Intelligence: Concepts, Techniques, and Applications in Microsoft Office Excel with XLMiner. John Wiley and Sons (2011).
19. Sangeetha, C., Amudha, P., Sivakumari, S.: Feature Extraction Approach For Spam Filtering. Int. J. Adv. Res. Technol. 2, 89–93 (2012).

20. Goweder, A.M., Rashed, T.E., Ali, S., Alhammi, H.A.: An Anti-spam system using artificial neural networks and genetic algorithms. Proc. 2008 Int. Arab Conf. Inf. Technol. 1–8 (2008).
21. Khorsi, A.: An Overview of Content-Based Spam Filtering Techniques. Inform. Slov. 31, 269–277 (2007).
22. Katirai, H.: Filtering Junk E-Mail: A Performance Comparison between Genetic Programming and Naive Bayes, http://citeseer.ist.psu.edu/310632.html, (1999).
23. Hirsch, L., Saeedi, M., Hirsch, R.: Evolving Rules for Document Classification. In: Keijzer, M., Tettamanzi, A., Collet, P., Hemert, J. van, and Tomassini, M. (eds.) Genetic Programming. pp. 85–95. Springer Berlin Heidelberg (2005).
24. Shengen, L., Xiaofei, N., Peiqi, L., Lin, W.: Generating New Features Using Genetic Programming to Detect Link Spam. In: Proceedings of the 2011 Fourth International Conference on Intelligent Computation Technology and Automation - Volume 01. pp. 135–138. IEEE Computer Society, Washington, DC, USA (2011).
25. Payne, T., Payne, T.: Learning Email Filtering Rules with Magi A Mail Agent Interface. Presented at the Department of Computing Science, University of Aberdeen (1994).
26. Davenport, G.F., Ryan, M.D., Rayward-Smith, V.J.: Rule Induction Using a Reverse Polish Representation. In: GECCO. pp. 990–995 (1999).
27. Lichman, M.: UCI Machine Learning Repository, Irvine, CA, University of California, School of Information and Computer Science (2013). http://archive.ics.uci.edu/ml.
28. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA. (1988)
29. Koza J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. A Bradford Book (1992).
30. Koza J.R.: Genetic Evolution And Co-Evolution Of Computer Programs. In: Artificial Life II. pp. 603–629. Addison-Wesley Publishing Company (1990).
31. Koza J.R., K.M.A.: Genetic Programming IV. Kluwer Academic Publishers (2003).
32. Downey, C.: Explorations in Parallel Linear Genetic Programming: A Thesis Submitted to the Victoria University of Wellington in Fulfilment of the Requirements for the Degree of Master of Science in Computer Science. Victoria University of Wellington (2011).
33. Downey, C., Zhang, M.: Parallel linear genetic programming. In: Proceedings of the 14th European conference on Genetic programming. pp. 178–189. Springer-Verlag, Berlin, Heidelberg (2011).
34. Abraham, A., Ramos, V.: Web usage mining using artificial ant colony clustering and linear genetic programming. In: The 2003 Congress on Evolutionary Computation, 2003. CEC '03. pp. 1384–1391 Vol.2 (2003).
35. Gandomi, A.H., Alavi, A.H., Sahab, M.G.: New formulation for compressive strength of CFRP confined concrete cylinders using linear genetic programming. Mater. Struct. 43, 963–983 (2009).
36. Guven, A.: Linear genetic programming for time-series modelling of daily flow rate. J. Earth Syst. Sci. 118, 137–146 (2009).
37. Song, D., Heywood, M.I., Zincir-Heywood, A.N.: A Linear Genetic Programming Approach to Intrusion Detection. In: Genetic and Evolutionary Computation — GECCO 2003. pp. 2325–2336. Springer Berlin Heidelberg (2003).
38. Mukkamala, S., Sung, A.H., Abraham, A.: Modeling Intrusion Detection Systems Using Linear Genetic Programming Approach. In: Orchard, B., Yang, C., and Ali, M. (eds.) Innovations in Applied Artificial Intelligence. pp. 633–642. Springer Berlin Heidelberg (2004).

39. Kononenko, I.: Semi-naive bayesian classifier. In: Kodratoff, Y. (ed.) Machine Learning — EWSL-91. pp. 206–219. Springer Berlin Heidelberg (1991).
40. Hamblin, C.L.: Translation to and from Polish Notation. Comput. J. 5, 210–213 (1962).
41. RPN, An Introduction To Reverse Polish Notation, http://h41111.www4.hp.com/calculators/uk/en/articles/rpn.html.
42. Burks, A.W., Don W. Warren, Wright, J.B.: An Analysis of a Logical Machine Using Parenthesis-Free Notation. Math. Tables Aids Comput. 8, 53–57 (1954).
43. galculator - a GTK 2 / GTK 3 algebraic and RPN calculator, http://galculator.sourceforge.net/.
44. Bennett, P.N.: Assessing the Calibration of Naive Bayes' Posterior Estimates. School of Computer Science, Carnegie Mellon University (2000).
45. Monti, S., Cooper, G.F.: A Bayesian Network Classifier that Combines a Finite Mixture Model and a Naive Bayes Model. arXiv:1301.6723. (2013).
46. Safe Browsing Tool | WOT (Web of Trust), http://www.mywot.com/.
47. Safe Browsing API — Google Developers, https://developers.google.com/safe-browsing/.
48. Damodaram, R., Valarmathi, D.M.L.: RBL Global Toolbar with Clustering Algorithm for Fake Website Detection.
49. Bennett, P.E.: The Statistical Measurement of a Stylistic Trait in Julius Caesar and As You Like It. Shakespeare Q. 8, 33–50 (1957).
50. Stamatatos, E., Fakotakis, N., Kokkinakis, G.: Computer-based Authorship Attribution without Lexical Measures. Comput. Humanit. 35, 193–214 (2001).
51. Yatsko, V.A.: Automatic text classification method based on Zipf's law. Autom. Doc. Math. Linguist. 49, 83–88 (2015).
52. Basavaraju, M., Prabhakar, D.R.: A novel method of spam mail detection using text based clustering approach. International Journal of Computer Applications. 5, 15–25 (2010).
53. Matsumoto, M., Nishimura, T.: Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. ACM Trans Model. Comput. Simul. 8, 3–30 (1998).
54. Pdnsd: pdnsd homepage, http://members.home.nl/p.a.rombouts/pdnsd/.
55. Jong, K.A.D., Spears, W.M.: An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms. In: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature. pp. 38–47. Springer-Verlag, London, UK, UK (1991).
56. Zhang, M., Ciesielski, V.: Genetic Programming for Multiple Class Object Detection. In: Foo, N. (ed.) Advanced Topics in Artificial Intelligence. pp. 180–192. Springer Berlin Heidelberg (1999).
57. Piszcz, A., Soule, T.: Genetic Programming: Analysis of Optimal Mutation Rates in a Problem with Varying Difficulty. In: FLAIRS Conference. pp. 451–456 (2006).
58. Cormack, G.V., Lynam, T.R.: Online supervised spam filter evaluation. ACM Trans Inf Syst. 25, 11 (2007).
59. Graham-Cumming, John: Understanding Spam Filter Accuracy (Newsletter), http://www.jgc.org/antispam/11162004-baafcd719ec31936296c1fb3d74d2cbd.pdf.
60. Claypool, Mark, O'Brien, Jason: An Analysis of Spam Filters. Computer Science Department, WPI (2003).