原著論文 Original Paper

Unsupervised Spam Detection by Document Probability Estimation with Maximal Overlap Method

Takashi Uemura — Hokkaido University
tue@ist.hokudai.ac.jp, http://www-ikn.hokudai.ac.jp/˜tue/

Daisuke Ikeda — Kyushu University
daisuke@i.kyushu-u.ac.jp, http://www.i.kyushu-u.ac.jp/˜daisuke/

Takuya Kida — Hokkaido University
kida@ist.hokudai.ac.jp, http://www-ikn.hokudai.ac.jp/˜kida/

Hiroki Arimura — (affiliation as previous author)
arim@ist.hokudai.ac.jp, http://www-ikn.hokudai.ac.jp/˜arim/

keywords: unsupervised spam detection, document complexity, suffix tree, maximal overlap method, word salad

Summary

In this paper, we study content-based spam detection for spams that are generated by copying a seed document with some random perturbations. We propose an unsupervised detection algorithm based on an entropy-like measure called document complexity, which reflects how many similar documents exist in the input collection of documents. As the document complexity, however, is an ideal measure like Kolmogorov complexity, we substitute an estimated occurrence probability of each document for its complexity. We also present an efficient algorithm that estimates the probabilities of all documents in the collection in linear time to its total length. Experimental results showed that our algorithm especially works well for word salad spams, which are believed to be difficult to detect automatically.

1. Introduction

Detecting and removing *spam messages* is a matter of considerable concern to everybody. Spams are ubiquitous in the diversified media of the Internet and we suffer great losses from them; more than 90 percent of all emails have been reported as spam [European 06], and more than 75 percent of all blog entries have also been reported as splog (spam-blog) [Kolari 06]. They wastefully consume network resources, unnecessarily lay a burden on server systems, and degrade the accuracy of search results. According to a report of European Commission [Postini 06], spam emails cause damage worths 39 billion euros worldwide. Some countries have placed legal restraints on sending spam messages, but we can not say that these restraints contribute to decrease the number of spam messages. Therefore, it is essential to develop an efficient method that detects spam automatically.

A lot of automatic spam detection methods have been proposed. In the early stages of the history, rule-based methods were widely used, such as compilation blacklists of IP addresses. Then machine-learning-based methods, such as a Bayesian method in [Graham 02], have been used; these methods learn probabilistic distributions of some features from given training examples and then judge a new coming email whether it is spam or non-spam, say *ham*, using the learned distributions. Since it is too costly to create the necessary training data, unsupervised methods have been proposed [Narisawa 06, Narisawa 07, Yoshida 04]. However, to circumvent these methods, spammers have created spam messages in more sophisticated ways, such as *word salads*. We have to develop a novel method for such a new type of spam.

Spam detection methods are categorized into two groups: non-content-based methods and content-based ones. A blacklist-based filtering is a typical example of the former, and the Bayesian method in [Graham 02] is the latter. The algorithms stated in [Narisawa 06, Narisawa 07], which find characteristic substrings in the input documents to recognize spam, are content-based methods, too. The algorithm in [Yoshida 04] judges some emails as spam if the number of similar emails is larger than a given threshold value. The algorithm in [Mishne 05] estimates language models of spam and ham messages. The algorithms stated in [Gyöngyi 04, Benczúr 05] utilize the hyperlink structure of contents. For word salads, there are a few reports. Misra *et al.* [Misra 08] present a method that measures the semantic coherence of a document by using Latent Dirichlet allocation. Their idea is based on the premise that a ham contains only a few topics but a spam is made up of many topics. Jeong *et al.* [Jeong 10] present a method that utilizes thesauruses, which regards a message as a spam when the distribution of terms in it differs from the normal distribution of the thesauruses.

In this paper, we study content-based spam detection methods. We assume that spam documents are automatically generated as copies of a seed document with allowing some random modifications [Bratko 06], since we can observe that the aim of spammers is to obtain large rewards with a slight effort and hence they should create a large number of copies. Conversely, the *cost* of generating a normal document should be relatively high.

The main contribution of this paper is to propose an unsupervised algorithm for spam detection, which we call *the spam detection algorithm by Document Complexity Estimation* (DCE for short). The key idea is to measure the cost of generating each document by an entropy-like measure, called *document complexity*. It is defined as the ideal average code-word-length of the document by presuming the input collection excluding the target itself. We expect that the complexity for ham is high, but low for spam. As the document complexity, however, is an ideal measure like Kolmogorov complexity, we substitute an estimated occurrence probability of each document for its complexity. We treat each document as a sequence of letters over a finite alphabet rather than a bag-of-words, and estimate its generation probability from a latent model of the input collection by using *Maximal Overlap method* proposed by Jagadish *et al.* [Jagadish 99].

Maximal Overlap method [Jagadish 99] (MO for short) is a technique for estimating the probability $P(x)$ of a long string x from the probabilities of substrings in x. Although a straightforward method based on the original MO takes more than quadratic time to the target string length, we present an efficient algorithm that runs in linear time to it. We call it as DCE by MO algorithm (DCE_MO for short). The keys of the algorithm is full exploitation of the suffix tree and suffix links.

We ran experiments on the real data from popular bulletin boards and some synthetic data. We compared our algorithm with the algorithm stated in [Narisawa 07], which is also an unsupervised method. For the real data, the results of DCE_MO are comparable to the algorithm in [Narisawa 07]. Experimental results on the synthetic data showed that DCE_MO particularly works well for spam messages based on random replacement, such as word salad and noise spam.

The rest of this paper is organized as follows. In Chapter 2 we reviews basic notions. In Chapter 3 we gives our spam detection algorithm. In Chapter 4 we reports experimental results. Finally, in Chapter 5 we conclude this paper.

The previous version of this paper appeared in [Uemura 08]. We revised the representation through the whole paper, and carried out several additional experiments. Since, in the experiments in Chapter 4, there were some bugs in our implementations of our method and [Narisawa 07], we fixed and reevaluated them; therefore, the results in this time are different from those in the previous paper.

2. Preliminaries

2·1 Strings

Let N be the set of all non-negative integers, and let Σ be a finite *alphabet*. We denote by Σ^* the set of all finite *strings* over Σ. The *length* of x is denoted by $|x| = n$. The *empty string* is the string of length zero and denoted by ε. For a string s, if $s = xyz \in \Sigma^*$ for some strings $x, y, z \in \Sigma^*$, then x, y, and z are called a *prefix*, a *substring*, and a *suffix* of s, respectively. If $s = a_1 \cdots a_n \in \Sigma^*$ $(a_i \in \Sigma)$ is a string of length n, then for every $1 \le i \le n$, the i-th letter of s is denoted by $s[i] = a_i$, and the *substring* from i to j is denoted by $s[i..j] = a_i \cdots a_j$; we define $s[i..j] = \varepsilon$ if $i > j$ for convenience. The *concatenation* of strings s and t is denoted by $s \cdot t$, or simply st. We say that a string *s occurs in string t at the position i* if $s[1..|s|] = t[i..i+|s|-1]$. For a set $D = \{s_1, \ldots, s_M\} \subseteq \Sigma^*$ $(M \ge 0)$ of M strings, we define $|D| = M$ and $||D|| = \sum_{s \in D} |s|$. We define the sets of *all substrings* and *all suffices* of D by $Sub(D) = \{\, s[i..j] : s \in D, 1 \le i, j \le |s| \,\}$ and $Suf(D) = \{\, s[i..|s|] : s \in D, 1 \le i \le |s| \,\}$, respectively.

2·2 Suffix Trees

Let $D = \{s_1, \ldots, s_M\}$ $(M \ge 0)$ be a set of M strings over Σ with the total size $N = ||D||$. The *suffix tree* for D, denoted by $ST(D)$, is a compacted trie $ST(D) = (V, E, root, suf, lab, fr)$ [Ukkonen 95] for the sets of all suffices of the strings in D as shown in Figure 1. Formally, the suffix tree for D is a rooted tree, where V is the vertex set, $E \subseteq V^2$ is the edge set, $suf : V \to V$ is a suffix link, $lab : E \to \Sigma^*$ is an edge-labeling, and $fr : V \to \mathsf{N}$ is a frequency counter. All the out-going edges leav-

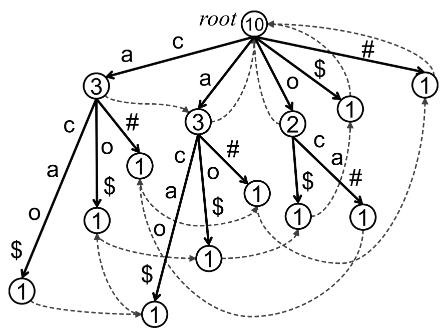

Fig. 1 The suffix tree for $D = \{cacao\$, oca\#\}$. Broken lines represent the suffix links. Figures in circles indicate the frequency counters.

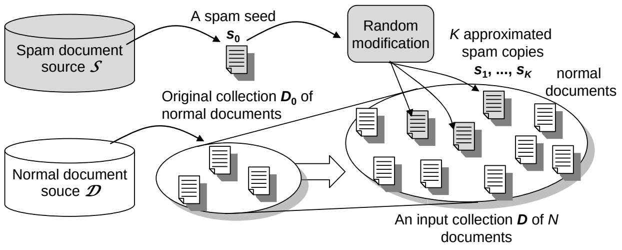
Spam document source $\mathcal{S}$
A spam seed
s_0
Random modification
K approximated spam copies
$s_1, ..., s_K$
normal documents
Original collection D_0 of normal documents
Normal document souce $\mathcal{D}$
An input collection D of N documents

Fig. 2 Blog and Bulletin Board Spam-Generation Process

ing from a vertex always start with mutually different letters. Each vertex $v \in V$ represents the unique substring $\alpha = [v] \in Sub(D)$, where $[v]$ is the concatenation of the labels on the unique path from the root to v. Therefore, we identify a vertex with the corresponding string hereafter. For every internal vertex $[c\alpha]$ starting with a symbol $c \in \Sigma$, there always exists a suffix link from $[c\alpha]$ to the unique vertex $suf([c\alpha]) = [\alpha]$. Finally, the leaves of $ST(D)$ represent the set $Suf(D)$, and thus, $ST(D)$ stores $Sub(D)$. In Figure 1, we show the suffix tree for a string $s = cacao\$$, where solid and broken lines represent the edges and the suffix links, respectively. Numbers in vertices represent their frequency counters. We note that \$ is the *end-marker* which does not occur in $s[1..|s|-1]$, and then $ST(s)$ strictly has $|s|$ leaves.

$ST(D)$ has at most $2N-1$ vertices since the common prefix of paths can be shared. Ukkonen [Ukkonen 95] presented an elegant algorithm that builds $ST(D)$ in $O(N)$ time by traversing the tree using suffix links. We augment each vertex $v = [\alpha]$ with the counter $fr(v) \in \mathsf{N}$, which keeps the frequency of α as a substring in strings of D. We can see that $fr([\alpha])$ equals the number of leaves that are descendants of $[\alpha]$.

2·3 *Blog and Bulletin Board Spam-Generation Process*

We focus on a type of spam called *blog* and *bulletin board spam*. Figure 2 shows an outline of a mechanism of spam generation for this type of spam.

We first assume a large collection D_0 consisting of normal documents randomly drawn from the document source $\mathcal{D}$. In typical situations, normal documents are posted by human users. Next we assume a document s_0 is drawn randomly as a *spam seed* from the spam document source $\mathcal{S}$ to generate a collection of spam documents. For an integer $K \geq 1$, which is unknown to the users, K approximate copies $s_1, \ldots, s_K$ of the seed s_0 are generated with allowing the following perturbations:

Mutation Changing randomly selected letters in a document.

Crossover Exchanging randomly selected parts of a pair of documents.

Word Salad Replacing a set of randomly selected words in a sentence with randomly selected words drawn from other sentences so that it is still grammatically correct, but meaningless.

Then, the spam documents $s_1, \ldots, s_K$ are added to D_0. The input collection of documents $D = (d_1, \ldots, d_n)$, $n \geq 0$ is generated by repeating this process for different spam seeds. Note that we can not see in advance how many spam seeds are selected and how many spam documents are generated. A document $d \in D$ is said to be *dirty* if it is spam, and *clean* if it belongs to the original collection D_0.

We note that our detection algorithm in Chapter 3 is *adaptive*, and thus it does not need a priori knowledge about the document sources $\mathcal{D}$ and $\mathcal{S}$, classes of modifications, and the number K of approximate copies per seed.

2·4 *Spam Detection Problem*

Given an input collection D, we consider the problem of classifying all documents in D into clean or dirty. Formally, our goal is to devise a mapping $f : D \to \{0, 1\}$ that makes classification, where the values 0 and 1 indicate the states that d is clean or dirty, respectively. We call the mapping *decision function*. We measure the *performance* of f on D by some cost functions. Let M_+ be the number of correctly detected spam documents, and M_- be the number of nomal documents judged as spam. Let also N_+ be the total number of spam documents in D. Then, the *recall* is defined by $R = M_+/N_+$ and the *precision* is defined by $P = M_+/(M_+ + M_-)$. The *F-score* is also defined by $F = 2PR/(P+R)$.

3. The Proposed Spam Detection Method

3·1 *Outline of Proposed Method*

Our method is parameterized by a given *probabilistic model* $\mathcal{M}$ for probability distribution over Σ^*. We assume the model $\mathcal{M} = \{\, Q(\cdot \mid \theta) : \theta \in \Theta \,\}$, which is a family of probability distributions over strings $Q(\cdot \mid \theta) : \Sigma^* \to [0,1]$ with parameter θ drawn from some parameter space Θ. A parameter θ can be a description of a non-parametric model, e.g., Markov chains, as well as a parametric model.

The basic idea of our method is as follows. Let $D \subseteq \Sigma^*$ be an input collection of M documents. Let $d \in D$ be any target document in D. If we model D by some probability distribution $Q(\cdot \mid \theta)$ for some $\theta \in \Theta$, then we can encode D by a certain encoding whose code length is

$$L(D \mid \theta) \simeq \lceil -\log Q(D \mid \theta) \rceil$$

so that the best compression ratio can be achieved [Cover 38]. In what follows, we denote by θ_D this best parameter θ for a given collection D within Θ. If d is *dirty*, then d is a copy of some spam seed $s_0 \in \mathcal{S}$ that has a certain number of copies in D. In this case, we expect that d is statistically similar to some portion of D, and thus, the contribution of d to the whole D will be very small.

On the other hand, we suppose that the set D is obtained from the collection $D' = D \backslash \{d\}$ of $M-1$ documents by adding the document d. D' can again be encoded by some $\theta' = \theta_{D'} \in \Theta$ in length $L(D' \mid \theta')$. Then, the contribution of d can be measured by how much this addition of d increases the code length of D, which is bounded above by

$$\begin{aligned}\Delta L(D,d) &\stackrel{\text{def}}{=} L(D' \cup \{d\} \mid \theta) - L(D' \mid \theta') \\ &= L(D \mid \theta) - L(D' \mid \theta') \;\le\; L(d \mid \theta'),\end{aligned}$$

where $\theta = \theta_D$ and $\theta' = \theta_{D'}$. In the above equations, the inequality in the last row follows from the following observation: A description of D can be obtained by appending to the description for D' of length $L(D' \mid \theta')$ an encoding for d of length $L(d \mid \theta')$ for d conditioned by $D' = D \backslash \{d\}$. Therefore, we have an inequality $L(D \mid \theta) \le L(D' \mid \theta') + L(d \mid \theta')$.

We expect that if the target document d has more copies in D then the contribution $\Delta L(D,d)$ will be smaller and thus more likely to be spam. Then, $\Delta L(D,d)$ can be estimated by the upperbound $\Delta L(D,d) \simeq L(d \mid \theta') \simeq \lceil -\log Q\ (d \mid \theta') \rceil$. This is our decision procedure for spam detection:

$$f(d) = \begin{cases} 1 & \text{if } \lceil -\log Q(d \mid \theta') \rceil / |d| \le \gamma \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $\gamma > 0$ is a threshold parameter, which will be given in the following section. We define the *document complexity* of d for D by $\lceil -\log Q(d \mid \theta') \rceil / |d|$.

```
Algorithm MO:
Input: Any data structure $Sub(D)$ for a set $D \subseteq \Sigma^*$ of input strings, frequency counter $fr : Sub(D) \to \mathsf{N}$, and a string $s \in \Sigma^*$ of length $L$.
Output: An estimated probability of $\tilde{Q}(s|\theta_D)$.
  (i) Let $\gamma_0 = \varepsilon$ be the context, $fr(\varepsilon) = ||D||$, $\theta_D = (Sub(D), fr)$, and $i := 2$.
  (ii) While $s \ne \varepsilon$ holds, we repeat the following process: Find the unique longest substring $\gamma_i \in Sub(D)$ that maximizes $\gamma_{i-1} \otimes \gamma_i$ such that $\gamma_i \otimes s \ne \varepsilon$. If there are more than two substrings with the same overlap, then take the longer one. In case that there exists no such $\gamma_i$, set $\alpha_i = \varepsilon$, $\beta_i = \gamma_i$ to be the initial letter of $s$, $P(\beta_i \mid \alpha_i) = 1/||D||$. Let $\alpha_i = \gamma_{i-1} \otimes \gamma_i$ and $\beta_i = \gamma_i \otimes s$. Remove the overlap $\gamma_i \otimes s$ from $s$.
       Define $P(\beta_i \mid \alpha_i) = fr(\alpha\beta_i)/fr(\alpha_i)$.
  (iii) Set $m = i$.
       Return $\tilde{Q}(s \mid \theta_D) = P(\beta_1) \prod_{i=2}^{m} P(\beta_i \mid \alpha_i)$.
```

Fig. 3 The original MO algorithm for estimating the probability $Q(s \mid \theta_D)$

3·2 *Document Probability Estimation by* MO *Method*

We did not mention about the concrete model that the probability of each document follows. In general, to compute the precise value of $Q(d \mid \theta')$, namely the document complexity of d for D, is quite hard. Therefore, we substitute an estimated occurrence probability of d for its complexity. We denote the substitute probability by $\tilde{Q}(d \mid \theta')$. We use Maximal Overlap method [Jagadish 99] (MO for short) to estimate $\tilde{Q}(d \mid \theta')$.

MO estimates the probability $P(x)$ of a long string $x \in \Sigma^*$ based on a set of shorter substrings appearing in the original string in D as follows. Given string β, the *conditional probability* for a string α is given by $P(\beta \mid \alpha) = P(\alpha\beta)/P(\alpha)$. Let $s \in \Sigma^*$ be a *target* string to estimate $\tilde{Q}(s \mid \theta_D)$. In general, for any $\tilde{Q}(\cdot \mid \theta)$, we can write the probability of the string $s = \beta_1 \cdots \beta_m$ $(m \ge 1)$ as $P(s) = P(\beta_1) \prod_{i=2}^{m} P(\beta_i \mid \beta_1 \cdots \beta_{i-1})$. For each $i = 1, \ldots, m$, MO approximates the *conditional probability* $P(\beta_i \mid \beta_1 \cdots \beta_{i-1})$ by the conditional probability $P(\beta_i \mid \alpha_i)$ with the unique longest suffix α_i of $\beta_1 \cdots \beta_{i-1}$, called the *longest context*, such that $\alpha_i\beta_i$ appears in D. For substrings u and v of an input string s, we define the *maximal overlap*, denoted by $u \otimes v$, as the maximal string $w \in \Sigma^*$ that is both a suffix of u and a prefix of v.

Recall that $Sub(D)$ is the set of substrings in D. In Figure 3, we show the original MO method that computes an estimation of $P(s)$ in the greedy way. For every $i =$

```
Algorithm LinearMO:
Input: A string $s$ and the suffix tree $ST(D)$for a set of strings $D$.
Output: an estimated probability of $\tilde{Q}(s|\theta_D)$.
 1: $v :=$ the root of $ST(D)$;
 2: for $i := 1, \ldots, \ell = |s|$ do
 3:     $x_i := s[i]$;
 4:     while $v$ has no out-going edge $(v, w)$ labelled with $x_i$ do
 5:         if $(v = root)$ then $Q := Q \cdot (1/N)$ with $N = ||D||$;
 6:         else $v := suf(v)$ by following the suffix link;
 7:     end while
 8:     $Q := Q \cdot fr(w)/fr(v)$;
            /* $\alpha_i = str(v)$ and $P(x_i|\alpha_i) = fr(w)/fr(v)$ */
 9:     $v := w$ by following the edge;
10: end for
11: return $Q$;  /* estimation of $\tilde{Q}(s|\theta_D)$ */
```

Fig. 4 A linear time MO-score estimation algorithm

$1, \ldots, |s|$ $(|s| \geq 1)$, MO finds maximally overlapping substrings $\gamma_i = \alpha_i\beta_i \in Sub(\{s\})$ such that $\beta_i \neq \varepsilon$. Therefore, we define the associated probabilities by $P(\beta_i \mid \alpha_i) = fr(\alpha_i\beta_i)/fr(\alpha_i)$. If there exists no such γ_i then the next letter $s[k] \in \Sigma$ has never appeared in D, where $k = |\beta_1 \cdots \beta_{i-1}|$. Then in this zero-probability case, we set $\alpha_i = \varepsilon$, $\gamma_i = \beta_i$ to be the un-seen letter $s[k] \in \Sigma$, and define $P(\beta_i \mid \alpha_i) = 1/N$, where $N = ||D||$ is the total letter frequency. Note that $s = \beta_1 \cdots \beta_{|s|}$. Then, we compute the probability by $\tilde{Q}(s \mid \theta_D) = P(\beta_1)\prod_{i=2}^{|s|} P(\beta_i \mid \alpha_i)$.

Jagadish *et al.* [Jagadish 99] show that MO is computable in $O(\ell q)$ time for given the suffix tree $ST(D)$, where $\ell = |s|$ and q is the depth of $ST(D)$. That is, MO takes $O(\ell^2)$ time in worst case. They also show that MO outperforms the previous method KVI [Krishnan 96], which uses the non-overlapping decomposition of the input string.

3·3 *Efficient Computation of* MO

In Figure 5, we show the outline of the proposed algorithm DCE_MO that runs in $O(N)$ to detect all spam candidates, where $N = ||D||$ is the total length of documents. The crucial parts of the algorithm are at Line 4 and 5. That is a time-comsuming process since we have to iteratively build the suffix tree M times for $\mathcal{D}$ except for the current document D to estimate $\tilde{Q}(d \mid \theta')$ for each $d \in D$. Therefore, a straightforward implementation requires $O(MN + M\ell^2) \simeq O(MN + N\ell) = O(MN)$ time, where $M = |D|$, $N = ||D||$, and $\ell = \max_{d \in D} |d|$.

Speed-up of MO estimatoin.

Figure 4 shows a linear-time algorithm LinearMO for estimating $Q(s|\theta_D)$ using $ST(D)$. The algorithm quickly

```
Algorithm DCE_MO($D$):
Input: An input collection $D \subseteq \mathcal{D}$, a threshold $\gamma > 0$.
Output: A set $A \subseteq D$ of dirty documents in $D$.
1: Construct the suffix tree $ST(D)$ for the whole document $D$;
2: $A := \emptyset$;
3: foreach $d \in D$ do
4:     Simulate $Q :=$ LinearMO$(d, ST(D \setminus \{d\}))$ by running
       copies of LinearMO simultaneously on $ST(D)$ and $ST(\{d\})$;
5:     $L := -\log Q$;
6:     if $L/|d| \leq \gamma$ then $A := A \cup \{d\}$;
           /* $d$ is detected as a spam */
7: end for
8: return $A$;
```

Fig. 5 An outline of our spam detection algorithm

finds the longest context α_i for each factor β_i by traversing the suffix tree using suffix links via a technique similar to [Ukkonen 95]. By the next lemma, LinearMO improves computation time of the original MO from $O(M\ell^2)$ time to $O(M\ell) \simeq O(N)$ time.

[Lemma 1] Let Σ be an alphabet of constant size. The algorithm LinearMO in Figure 4 correctly implements the algorithm MO in Figure 3 to estimate $Q(s \mid \theta_D)$, and runs in $O(\ell)$ time and $O(N)$ space, where $N = ||D||$ is the total size of the sample D and ℓ is the length of the string s.

《**Proof**》 Let $\gamma_i = \alpha_i\beta_i$ $(m \geq 1$ and $1 \leq i \leq m)$ be the maximally overlapping substrings computed in the original MO algorithm. We assume that $\beta_j = x_h \cdots x_i$ $(h \leq i)$ with $k = |\beta_i|$. Then, we can show that $P(\beta_i \mid \alpha_i) = P(x_h \mid \alpha_i)P(x_{h+1} \mid \alpha_i x_h) \cdots P(x_i \mid \alpha_i x_h \cdots x_{i-1})$ because the longest context of x_j in s relative to $ST(D)$ is the string $\alpha x_h \cdots x_{j-1}$ for each $h \leq j \leq i$. Thus, the correctness follows. The time complexity is derived from arguments similar to [Ukkonen 95].

Avoiding repetitive construction of suffix trees.

In the building phase at Line 4 of Algorithm DCE_MO, a straightforward implementation takes $O(MN + M\ell^2)$ time by building the suffix tree $ST(D \backslash \{d\})$, called a *leave-one-out suffix tree* for D, for each document $d \in D$. Instead of this, we simulate the traversal of the leave-one-out suffix tree $ST(D \backslash \{d\})$ directly on $ST(D)$ for D and $ST(\{d\})$. A *virtual leave-one-out suffix tree* for D and d is a suffix tree $\widetilde{ST}(D \setminus \{d\}) = (V', E', root', suf', lab', fr')$ defined by $V' = \{\, [\alpha] : \alpha \in \Sigma, fr_D(\alpha) \geq 1, fr_d(\alpha) \geq 1 \,\}$ and $E' = \{\, ([\alpha], [\beta]) \in V' \times V' : ([\alpha], [\beta]) \in E_D \,\}$, where $fr_D(\alpha)$ and $fr_d(\alpha)$ are the frequency counters of α on $ST(D)$ and $ST(d)$, respectively. We define the frequency of a vertex $[\alpha]$ by $fr'([\alpha]) = fr_D([\alpha]) - fr_d([\alpha])$.

[Lemma 2] Let $D \subseteq \Sigma^*$ and $d \in \Sigma^*$ be any collection and a document. Suppose that $d \in D$. Then, $\widetilde{ST}(D\backslash\{d\})$ is isomorphic to the original $ST(D\backslash\{d\})$.

[Lemma 3] Let $D \subseteq \Sigma^*$ be any sample. For any document $d \in D$ of length ℓ, we can simulate LinearMO in Figure 4 over the virtual suffix tree $\widetilde{ST}(D\backslash\{d\})$ by runnning the copies of LinearMO simultaneously over $ST(D)$ and $ST(\{d\})$ by the above rule. The obtained algorithm runs in $O(\ell)$ time with preprocess $O(||D|| + \ell)$ for constructing $ST(D)$ and $ST(\{d\})$.

《**Proof**》 The algorithm LinearMO has a pair (v_D, v_d) of two pointers $v_D \in V_D$ and $v_d \in V_d$, respectively, on vertices of $ST(D)$ and $ST(\{d\})$. When the algorithm receives the next letter $c = s[i]$, it moves from a combined state (v_D, v_d) to (v'_D, v'_d) such that $v'_D = [str_D(v_D) \cdot c]$ and $v'_d = [str_d(v_d) \cdot c]$ if $fr_D(v'_D) - fr_d(v'_d) \geq 1$ hold. For the zero-probability case (at line 5), we set $N := ||D|| - |d|$. To see that the moves are always well-defined, we define the pair for v by $\pi(\alpha) = (\mathbf{sgn}\, fr_D(\alpha), \mathbf{sgn}\, fr_d(\alpha)) \in \{+, 0\} \times \{+, 0\}$ for any vertex $v = [\alpha]$ $(\alpha \in \Sigma^*)$. Since $d \in D$ and $\alpha \in Sub(d)$ hold by assumption, Then, we can show that $(+, +)$ is only possible state for $\pi(\alpha)$. Hence,the result immediately follows.

From Lemma 1 and Lemma 3, we show the main theorem of this paper.

[Theorem 1] The algorithm DCE_MO in Figure 5 runs in $O(N)$ time and $O(N)$ space, where $N = ||D||$ is the total size of D.

3·4 Determining the Threshold

The method DCE_MO in Figure 5 takes a decision threshold $\gamma > 0$ as a learning parameter. To determine an appropriate value of γ, we first draw the histogram of the document complexity L over unlabelled examples only, and then adaptively define γ to be the point that takes the minimal value in an appropriate range, say, $0.0 \sim 1.0$ (bit).

To justify this selection of γ, we show in Figure 6 the histograms of the document complexity L over all documents in test collection YF4314, which we will state later. In the figure, we see that the distribution of spam documents is concentrated within the range of $L = 0.0 \sim 0.2$ (bit) per letter, while that of normal documents spreads over $L = 1.0 \sim 3.5$ (bit) per letter and obeys a bell-shaped distribution with the peak around 2.2 (bit). Thus, it is a straightforward strategy to minimize classification risk by taking γ to be the point that takes the minimal value. This justifies the choice of γ.

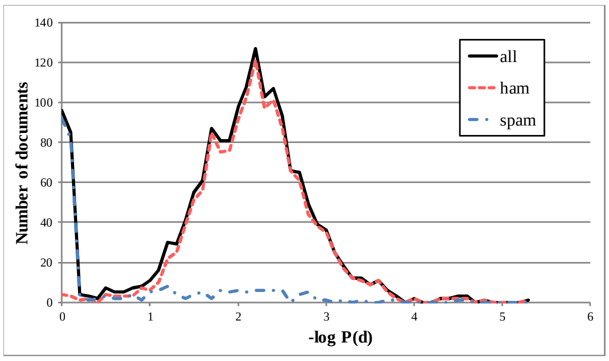

Fig. 6　Histogram of the document complexity of Web data.

4. Experimental Results

4·1 Dataset

We used a test collection of forums from Yahoo Japan Finance[*1], collected by Narisawa *et al.* [Narisawa 07]. The collection consists of four sections of forum data: YF4314, YF4974, YF6830, and YF8473. All posts of each forum are labelled if they are spam or not. Table 1 shows the details of them. In the following experiments, we used only the body texts , including HTML tags.

4·2 Method

For simplicity, we call the proposed algorithm by DCE from now on. We implemented DCE and the following methods.

Naive method: Let D be the input document set. Then, the Naive method regards a document d as spam if d is a substring of another document in D. That is, Naive is a type of copy-detection method.

Alienness measure: Narisawa *et al.* [Narisawa 07] propose a spam detection method which uses the *representatives* of substring classes, which are characteristic strings extracted from the document set. There are three measures for representatives, *length*, *size*, and *maximin*, and we call the methods with them the AM_Len, the AM_Siz, and the AM_Max, respectively. They also propose a method for determining the threshold for their measures. Their methods then regard a document as spam if it contains a representative such as alien, which is an outlier of substrings in nomal documents.

4·3 Results

Exp. 1: Computational efficiency.

First, we compared the implementation of LinearMO with a straightforward implementation of MO. The input document set D for this experiment is a concatenation of

*1 http://quote.yahoo.co.jp

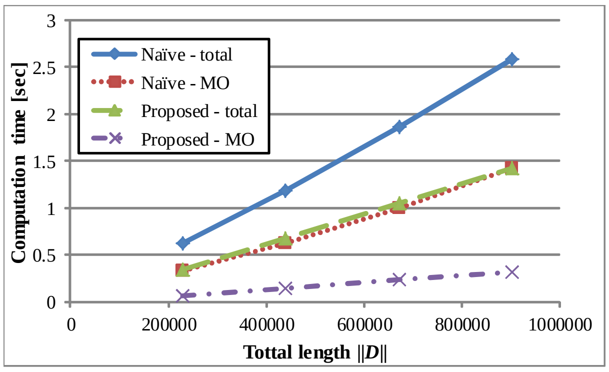

Fig. 7 Computation time.

Table 1 Details of the datasets.

Dataset	# of spam	# of ham	Total length
YF4314	291	1424	184,775
YF4974	331	1315	211,505
YF6830	317	1613	252,324
YF8473	264	1597	239,756

all the datasets. The total length $||D||$ is 888360. Figure 7 shows the computation time for the whole spam detection and for the probability estimation. As shown in the graph, the computation time for LinearMO is more than four times faster than the straightforward implementation. On the other hand, the total computation time is at most twice faster than that because the construction for suffix trees takes much time. The memory space consumption of the suffix tree of the whole D is about 29.7MB, including D itself.

Exp. 2: Basic Performance.

Table 2 shows the recalls, precisions, and F-scores for all methods. As shown in the table, DCE and AM_Siz achieved a high recall, DCE and Naive achieved high precision, and DCE achieved the highest F-score in all the datasets. Overall, DCE shows slightly better performance compared to other methods.

Exp. 3: Recall and Precision.

In this experiment, we did not use the methods to determine thresholds of each method. Instead, we output all documents as spam by ascending order of its document complexity and computed recalls and precisions. The graph in Figure 8 shows the recall and the precision curves of the dataset YF6830. To the left of the intersection of the precision and the recall of DCE, precision and recall are clearly higher than for any other methods. However, to the right of the intersection, the precision decreases faster than for the other methods.

Table 2 Performance of spam detection methods.

Forum	Method	Recall	Precision	F-score
4314	DCE	0.59	0.95	0.73
	AM_Len	0.54	0.92	0.68
	AM_Siz	0.82	0.73	0.78
	AM_Max	0.67	0.84	0.75
	Naive	0.54	0.94	0.68
4974	DCE	0.63	0.69	0.66
	AM_Len	0.4	0.72	0.51
	AM_Siz	0.78	0.63	0.69
	AM_Max	0.39	0.70	0.50
	Naive	0.52	0.66	0.58
6830	DCE	0.68	0.67	0.67
	AM_Len	0.45	0.63	0.52
	AM_Siz	0.81	0.48	0.61
	AM_Max	0.40	0.71	0.51
	Naive	0.54	0.71	0.62
8473	DCE	0.63	0.69	0.66
	AM_Len	0.54	0.72	0.62
	AM_Siz	0.81	0.52	0.63
	AM_Max	0.66	0.72	0.69
	Naive	0.54	0.79	0.64

Exp. 4: Performance Against Noise Density.

We created *noise spam* by using a seed as follows: For each position $1 \le i \le |d|$ of d, we replaced $d[i]$ by $d[j]$ with probability p, where $1 \le j \le |d|$ is a random number. We call the probability the *density* of noise. To examine the effect of the density, we set the number of spams to 20. Figure 9 shows the recall rates against density. In Exp. 4 and 5, we redefine the recall by (detected noise spams / inserted noise spams). AM_Siz was the best in the five algorithms. DCE were comparable to AM_Siz when $p \le 0.005$. However, its recall decreased depending on the increase of the density.

Exp. 5: Performance Against the Number of Spams.

In this experiment, we created noise spams by the same way as Exp. 4 with the density $p = 0.02$. Figure 10 shows the performance against the number m of inserted noise spams. AM_Siz was the best when $m \le 20$, and DCE was the best when $m > 20$. According to the increase in the number of inserted spams, the performance of DCE showed improvement while the other methods did not improve.

Exp. 6: Detecting Word Salads.

Spammers create word salads by replacing words in a *skeleton* document with some *keywords*. In this experiment, we created word salads and inserted them into the document set. The dataset used was YF4974. We created the keyword set by manually selecting from the dataset

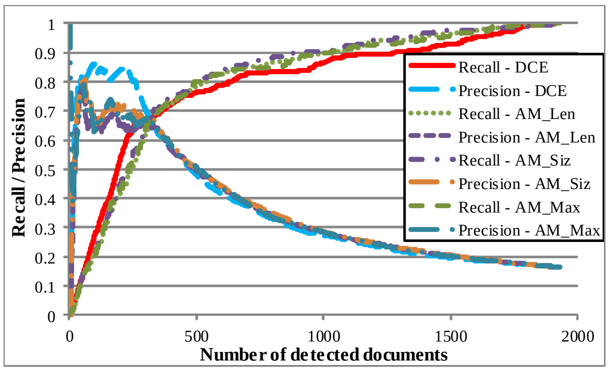

Fig. 8 Recall and Precision curve.

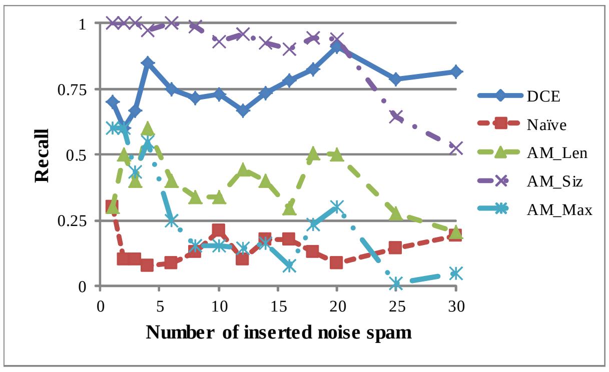

Fig. 10 Recall vs. number of inserted noise spams.

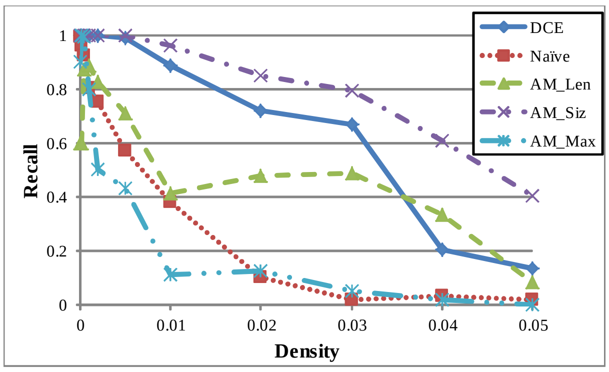

Fig. 9 Recall vs. density.

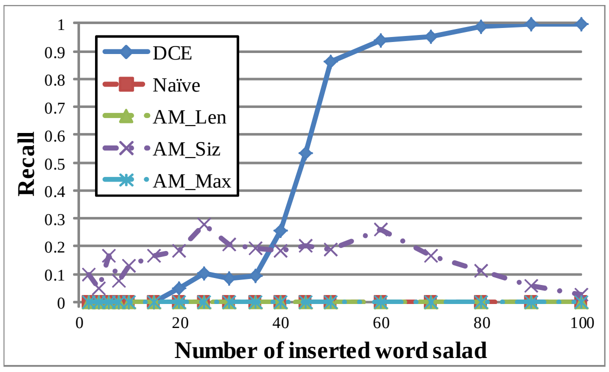

Fig. 11 Recall vs. number of inserted word salads.

and selected a spam as the skeleton in YF8473. The keyword set consisted of 25 nouns, and the skeleton consisted of 45 words. All nouns in the skeleton were replaced in the creation of the word salads. We then created these word salads and insert them into the dataset. Figure 11 shows the result. As well as Exp. 4 and 5, we define the recall by (detected word salads / inserted word salads). AM_Siz detected about 20% of word salads despite only a few word salads being inserted. However, the recall increased up to only 30%. DCE detected more word salads when the number m of word salads inserted was greater than 35. The recall was approximately 100% when $m > 60$. Since Naive only detects exact copies, it could not detect word salads. AM_Len and AM_Max could not detect word salads because their threshold values were relatively higher than AM_Siz.

We also show the precision in Figure 12. In this experiment, we define the pcrecision by (number of original spams and word salads / number of reported documents). Since the threshold value of each method did not change significantly, the number of detected original spams was almost same as the experiment of Exp. 2. DCE and AM_Siz slightly improved because they could detect word salads.

5. Conclusion

In this paper, we presented an unsupervised spam detection algorithm that calculates document complexity. The algorithm runs in linear time to the total length of the input documents. We also presented some experiment results for real and synthetic data; the real data were collected from popular bulletin boards and the synthetic data were generated artificially as word salad spam documents.

From the results of Exp. 6, it revealed that the performance of our method improves as the number of word salads increase, namely, as the cost of creating spams relatively goes down. Therefore we conclude that our method works successfully for such low complexity spams. As reported in [Sato 08], the ratio of splogs based on word salad to the whole splogs may be still low. However, it is always important to develop countermeasures against new and complicated spams in advance, since that spams which are hard to detect by existent methods, include query keywords, will increase rapidly on the Internet.

Very recently, Qian *et al.* [Qian 10] propose an unsupervised spam detection method, which has high performance to the level of state-of-the-art supervised methods. It is our future works to compare our method to such new methods

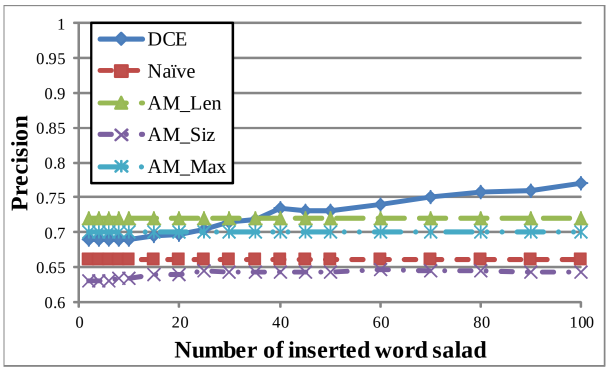

Fig. 12 Precision vs. number of inserted word salads.

[Qian 10, Jeong 10, Misra 08].

◇ References ◇

[Benczúr 05] Benczúr, A. A., Csalogány, K., and Tamás Sarlós, M. U.: SpamRank – Fully Automatic Link Spam Detection, in *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)* (2005), `http://airweb.cse.lehigh.edu/2005/`

[Bratko 06] Bratko, A., Filipič, B., Cormack, G. V., Lynam, T. R., and Zupan, B.: Spam Filtering Using Statistical Data Compression Models, *J. Mach. Learn. Res.*, Vol. 7, pp. 2673–2698 (2006)

[Cover 38] Cover, T. M. and Thomas, J. A.: *Elements of Information Theory*, Wiley, 2nd edition (1938)

[European 06] European, and Commission, : Fighting spam, spyware and malicious software: Member States should do better, says Commission, `http://europa.eu/rapid/pressReleasesAction.do?reference=IP/06/1629` (2006)

[Graham 02] Graham, P.: A Plan for Spam, `http://paulgraham.com/spam.html` (2002)

[Gyöngyi 04] Gyöngyi, Z., Garcia-Molina, H., and Pedersen, J.: COmbating Web Spam with TrustRank, in *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pp. 576–587 (2004)

[Jagadish 99] Jagadish, H. V., Ng, R. T., and Srivastava, D.: Substring selectivity estimation, in *Proc. of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems(PODS '99)*, pp. 249–260, ACM (1999)

[Jeong 10] Jeong, O.-R., Kim, W., Um, B.-Y., Kwon, J.-G., and Park, S.-H.: A word-salad filtering algorithm, *Logic Journal of IGPL* (2010)

[Kolari 06] Kolari, P., Java, A., and Finin, T.: Characterizing the splogosphere, in *Proceedings of the 3rd Annual Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics* (2006)

[Krishnan 96] Krishnan, P., Vitter, J. S., and Iyer, B.: Estimating alphanumeric selectivity in the presence of wildcards, in *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pp. 282–293, New York, NY, USA (1996), ACM

[Mishne 05] Mishne, G., Carmel, D., and Lempel, R.: Blocking Blog Spam with Language Model Disagreement, in *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)* (2005), `http://airweb.cse.lehigh.edu/2005/`

[Misra 08] Misra, H., Cappé, O., and Yvon, F.: Using LDA to detect semantically incoherent documents, in *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08, pp. 41–48, Morristown, NJ, USA (2008), Association for Computational Linguistics

[Narisawa 06] Narisawa, K., Yamada, Y., Ikeda, D., and Takeda, M.: Detecting Blog Spams using the Vocabulary Size of All Substrings in Their Copies, in *Proceedings of the 3rd Annual Workshop on Weblogging Ecosystem* (2006), `http://airweb.cse.lehigh.edu/2005/`

[Narisawa 07] Narisawa, K., Bannai, H., Hatano, K., and Takeda, M.: Unsupervised Spam Detection Based on String Alienness Measures, in *Discovery Science*, pp. 161–172 (2007)

[Postini 06] Postini, : Postini Announces Top Five 2007 Messaging Security Predictions as Email Spam Becomes Front Burner Issue Aagin in the New Year, `http://www.postini.com/news_events/pr/pr120606.php` (2006)

[Qian 10] Qian, F., Pathak, A., Hu, Y. C., Mao, Z. M., and Xie, Y.: A case for unsupervised-learning-based spam filtering, *SIGMETRICS Perform. Eval. Rev.*, Vol. 38, pp. 367–368 (2010)

[Sato 08] Sato, Y., Utsuro, T., Murakami, Y., Fukuhara, T., Nakagawa, H., Kawada, Y., and Kando, N.: Analysing features of Japanese splogs and characteristics of keywords, in *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, AIRWeb '08, pp. 33–40, New York, NY, USA (2008), ACM

[Uemura 08] Uemura, T., Ikeda, D., and Arimura, H.: Unsupervised Spam Detection by Document Complexity Estimation, in *DS '08: Proceedings of the 11th International Conference on Discovery Science*, pp. 319–331, Berlin, Heidelberg (2008), Springer-Verlag

[Ukkonen 95] Ukkonen, E.: On-line construction of suffix-trees, *Algorithmica*, Vol. 14, No. 3, pp. 249–260 (1995)

[Yoshida 04] Yoshida, K., Adachi, F., Washio, T., Motoda, H., Homma, T., Nakashima, A., Fujikawa, H., and Yamazaki, K.: Density-based spam detector, in *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 486–493, New York, NY, USA (2004), ACM

〔担当委員：鷲尾　隆〕

Received September 7, 2010.

Author's Profile

Takashi, Uemura

received B.S. in Engineering from Hokkaido University, Sapporo, Japan, in 2006. He recieved M.S. in Computer Science from Hokkaido University, Sapporo, Japan, in 2008. He is currently a doctral student of Division of Computer Science, Graduate School of Information Science and Technology, Hokkaido University. His research interests include text indexing, text mining, and data compression. He is a student member of The Institute of Electronics, Information and Communication Engineers.

Daisuke, Ikeda

Daisuke Ikeda is an associate professor in the Department of Informatics, Graduate School of Information Science and Electrical Engineering, Kyushu University. He received his B.S., M.S. and Ph.D. from Kyushu University in 1994, 1996, and 2004, respectively. He became an associate professor in Kyushu University in 2004. His current interests include data/text/Web mining, digital library, and academic communication infrastructure. He is a member of Information Processing Society of Japan, Association for Computing Machinery, and European Association for Theoretical Computer Science.

Takuya, Kida

received B.S. in Physics from Kyushu University, Fukuoka, Japan, in 1997. He received M.S. and Ph.D. in Computer Science from Kyushu University, in 1999 and in 2001, respectively. He was a full-time lecturer of Kyushu University Library from October 2001 to March 2004. Currently, he is an associate professor of Division of Computer Science, Graduate School of Information Science and Technology, Hokkaido University, since April 2004. His research interests include pattern matching algorithms, data compressions, and information retrieval. He is a member of The Institute of Electronics, Information and Communication Engineers, Information Processing Society of Japan and The Database Society of Japan.

Hiroki, Arimura (Member)

received the B.S. degree in 1988 in Physics, the M.S. and the Dr.Sci. degrees in 1990 and 1994 in Information Systems from Kyushu University. From 1990 to 1996, he was at the Department of Artificial Inteligence in Kyushu Institute of Technology, and from 1996 to 2004, he was at the Department of Informatics in Kyushu University. Since 2006, he has been a professor of the Division of Computer Science Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan. He has also been an adjunctive researcher with PREST program "Sakigake" of Japanese Science and Technology Agency for 1999 to 2002. His research Interests include data mining, computational learning theory, information retrieval, artificial intelligence, and the design and analysis of algorithms in these fields. He is a member of The Japanese Society for Artificial Intelligence, Information Processing Society of Japan, and Association for Computing Machinery.