Unsupervised Spam Detection by Document Complexity Estimation

Takashi Uemura[1], Daisuke Ikeda[2], and Hiroki Arimura[1]

[1] Hokkaido University, Kita 14, Nishi 9, Kita-ku, Sapporo, 060-0814 Japan
[2] Kyushu University, 744 Motooka Nishi-ku, Fukuoka 819-0395, Japan

Abstract. In this paper, we study a content-based spam detection for a specific type of spams called *blog* and *bulletin board spams*. We develop an efficient unsupervised algorithm DCE that, detects spam documents from a mixture of spam and non-spam documents using a compression-based similarity measure, called the *document complexity*. Using suffix trees, the algorithm computes the document complexity for all documents in linear time w.r.t. the total length N of input documents. Experimental results showed that our algorithm especially works well for detecting word salad spams, which are believed to be difficult to detect automatically.

1 Introduction

Spam messages are ubiquitous in the diversified media of the Internet. Recently, more than 90 percent of all emails have been reported as spam. As new communication media, such as blogs, have appeared, new types of spam messages adapted for these media have also arisen. For instance, more than 75 percent of all blog entries are blog spams, called splogs [1]. Spam messages cause great damage to our infrastructures: They wastefully consume network resources, unnecessarily burden the server systems of target media, such as blog servers, and degrade the accuracy of search results. According to a report, spam emails cause damage worths 39 billion euros world-wide. — it is therefore essential to develop methods that can detect and remove spam messages automatically.

A large number of automatic spam detection methods have been developed. In the early stages of the history of spam detection, rule-based methods were used, such as the compilation blacklists of IP addresses. Since then, machine learning-based methods have been used, e.g., a Bayesian method in [2]. In general, these methods learn probabilistic distributions of features from given training examples and then judge a new coming email on whether it is *spam* or non-spam, say *ham*, using these learned distributions. Since it is too costly to create the necessary training data, unsupervised methods were proposed [3–5]. However, to circumvent detection through these methods, spammers created spam messages in more sophisticated ways, such as image spam and word salad.

Spam detection methods are categorized into two groups: non-content-based methods and content-based ones. A method based on a blacklist is a typical example of the former. A Bayesian method outlined in [2] is a content-based method. Algorithms in [3, 4] find characteristic substrings in spam messages and so they are also content-based method. The algorithm in [5] judges some emails as spam if the number of similar emails is larger than a given threshold value. The algorithm in [6] estimates language models of spam and ham messages. Hyperlinks form part of content, and algorithms in [7, 8] make full use of their link structures.

In this paper, we study efficient content-based spam detection methods that can work with a large collection of input documents. We assume that spam documents are generated as copies of seed documents allowing some random operations [9] by reflecting the observation that the aim of spammers is to obtain large rewards with little effort and they have to create a large number of copies.

The main contribution of this paper is to propose an unsupervised algorithm for spam detection. The basic idea is to measure the *cost* of documents d relative to an input document collection D by an entropy-like measure $L(d \mid \theta_D)$, called *document complexity.* We expect that if documents are normal (ham) then its generation costs are sufficiently high. On the other hand, if they are spam document generated from seed documents, then their document complexity is quite low. The model parameter θ_D is computed by the suffix tree data structure, and the document complexity is estimated by an extension of a string probability model called MO method proposed by Jagadish *et al.* [12] based on the *leave-one-out suffix trees*.

However, a straightforward method based on the original MO takes almost quadratic time in the total input size N. To overcome this difficulty, by extending MO, we develop an efficient algorithm DCE (Algorithm for Document Complexity Estimation) that computes the document complexity for all documents d together with all leave-one-out suffix trees in in linear time in N. 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 synthetic data. For the real data, the results of our algorithm are comparable to the previous unsupervised algorithm in [4]. Experimental results on the synthetic data showed that our algorithm particularly works well for those spams such as *word salad spams* based on random replacement of inconsecutive regions. Organization of this paper is as follows. Section 2 reviews basic notions. Section 3 gives our spam detection algorithm DCE, and Section 4 reports experimental results, and Section 5 conclude this paper.

2 Preliminaries

2.1 Strings

Let $\mathsf{N} = \{0, 1, 2, \ldots\}$ and let Σ be a finite *alphabet*. We denote by Σ^* the set of all finite *strings* over Σ. by ε the *empty string* If $s = a_1 \cdots a_n \in \Sigma^*$ $(a_i \in \Sigma)$ is a string of length $n = |s|$, then for every $1 \leq i \leq n$, the i-th letter of s is $s[i] = a_i$, and the *substring* from i to j is $s[i..j] = a_i \cdots a_j$ if $i \leq j$ and $s[i..j] = \varepsilon$ otherwise. The *concatenation* of strings s and t is denoted by $s \cdot t$ or st. For a string s, if $s = xyz \in \Sigma^*$ for some strings $x, y, z \in \Sigma^*$ then x, y, and z are called, resp., a *prefix*, a *substring*, and a *suffix* of s. For a set $D = \{s_1, \ldots, s_m\} \subseteq \Sigma^*$ $(m \geq 0)$ of strings, we define $|D| = m$ and $||D|| = \sum_{s \in D} |s|$. We define the sets of *all*

Fig. 1. The suffix tree for a string $s = cacao\$$

Fig. 2. Blog and Bulletin Board Spam-Generation Process

substrings and *all suffices* of D by $Sub(D) = \{\ s[i..j] : s \in D, 1 \leq i \leq |s|\ \}$ and $Suf(D) = \{\ s[i..|s|] : s \in D, 1 \leq i \leq |s|\ \}$, resp.

2.2 Suffix Trees

Let $D = \{s_1, \ldots, s_m\}$ $(m \geq 0)$ be a set of strings over Σ with total size $N = ||D||$. Then, the *suffix tree* for D, denoted by $ST(D)$, is a compacted trie $ST(D) = (V, E, root, suf, lab, fr)$ [10] for the sets of all suffices of the strings in D as shown in Fig. 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 leaving from a vertex always start with mutually different letters. Each vertex $v \in V$ represents the unique substring $\alpha = str(v) \in Sub(D)$, where $str(v)$ is the concatenation of the labels on the unique path from the root to v. 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 Fig. 1, we show the suffix tree for a string $s = cacao\$$, where solid and broken lines represent the edges and the suffix links, respectively.

$ST(D)$ has at most $2N - 1$ vertices since the common prefix of paths can be shared [10]. Ukkonen [10] presented an elegant algorithm that build $ST(D)$ 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 show that $fr([\alpha])$ equals the number of leaves that are descendants of $[\alpha]$.

2.3 Blog and Bulletin Board Spam-Generation Process

In this paper, we focus on a specific type of spams called *blog* and *bulletin board spams*. Fig. 2 shows an outline of a spam-generating mechanism for this type of spam that is assumed in this paper. We 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.

On the other hand, to generate spam documents, we first randomly draw a document s_0, called a *spam seed*, from the source $\mathcal{S}$ for spam documents. For some appropriately chosen number $K \geq 1$, we then generate K approximate copies $s_1, \ldots, s_K$ of the original seed s_0 by applying the following perturbations to s_0 that are common to blog spams and bulletin board: **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 to have a grammatically correct, meaningless sentence.

Then, we add these K generated spam documents $s_1, \ldots, s_K$ to the original document set D_0. Repeating this process for different spam seeds, we create a mixture of many normal and some spam documents, called an *collection of input documents* $D = (d_1, \ldots, d_n),\ n \geq 0$. A document $d \in D$ is *dirty* if it is a spam and *clean* if it belongs to the original collection D_0.

Although we have described a hypothetical spam-generation process, we note that our detection algorithm in Section 3 is *adaptive* and thus does not need to know a priori the characteristics of probabilistic document sources $\mathcal{D}$ and $\mathcal{S}$, classes of modifications, or the number K of approximate copies per seed.

2.4 The Spam Detection Problem

We consider the following problem: given an input collection D of normal and spam documents, to classify all documents into clean or dirty. Our goal is to devise a mapping $f : D \to \{1, 0\}$, called a *decision function*, which makes classification, where values 1 and 0 indicates the states that d is dirty and d not, respectively. We measure the *performance* of f on D by some cost functions. Let $N = |D|$ and let M_+ (or M_-) be the number of detected spam (normal) documents, and N_+ $(N_-,)$ be the total number of spam (normal) documents in D. Then, the *recall* is $R = M_+/N_+$ and the *precision* is $P = M_+/(M_+ + M_-)$. The *F-score* is defined by $F = 2PR/(P + R)$.

3 The Proposed Spam Detection Method

In this section, we describe our proposed method for blog and bulletin board spam detection, called *DCE* (Document Complexity Estimation).

3.1 Outline of Our Decision Method

In our framework, each *document* is represented as a *sequence of letters* over an alphabet Σ rather than a *bag-of-words*. Our method is parameterized by a given probabilistic model $\mathcal{M}$ for probability distribution over Σ^*. We assume a *probabilistic 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 parameters 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.

In Fig. 3, we show the algorithm DCE1, which is an algorithmic schema of our method DCE. The basic idea of our method is as follows. Let $D \subseteq \Sigma^*$ be an input collection of M documents or a *sample*. If a given document $d \in D$ is *dirty* in D then d is a copy of some spam seed $s_0 \in \mathcal{S}$ that has a certain number of

```
Algorithm DCE1($D$):
Input: An input collection $D \subseteq \mathcal{D}$ of documents, a threshold $\gamma$
Output: A set $A \subseteq D$ of dirty documents in $D$.
1: $A := \emptyset$;
2: foreach $d \in D$ do
3:     $D' := D \backslash \{d\}$;
4:     Build a model $\theta' = \theta_{D'} \in \Theta$ for $D'$ by minimizing $L(D' \mid \theta')$;
5:     Estimate $L := L(d \mid \theta') \simeq \lceil -\log Q(d \mid \theta') \rceil$;
6:     if $L/|d| \leq \gamma$ then
7:         $A := A \cup \{d\}$; //$d$ is detected as a spam.
8: end for
9: return $A$;
```

Fig. 3. An outline of our spam detection algorithm

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 of D will be small. Let $d \in D$ be any target document in the sample D. If we model D by some probability distribution $Q(\cdot \mid \theta)$ for some $\theta \in \Theta$, then we can encode D in an encoding of the code length

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

so that best compression can be achieved [11]. In what follows, we denote by θ_D this best parameter θ for a given collection D within Θ.

On the other hand, we suppose that the sample D of M documents 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 the sample D, which is bounded above by

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

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 $D' = D \backslash \{d\}$. Hence, we have an inequality $L(D \mid \theta) \leq L(D' \mid \theta') + L(d \mid \theta')$.

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:

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

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. *Onput* an estimated probability of $Q(s|\theta_D)$.

```
(i) Initially, let $\gamma_0 = \varepsilon$ be the empty context and $fr(\varepsilon) = ||D||$. Let $\theta_D = (Sub(D), fr)$
    Let $i := 2$.
(ii) While $s \neq \varepsilon$ holds, we repeat the following process: Find the unique longest sub-
    string $\gamma_i \in Sub(D)$ that maximizes $\gamma_{i-1} \otimes \gamma_i$ such that $\gamma_i \otimes s \neq \varepsilon$. 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 $Q(s \mid \theta_D) = P(\beta_1) \prod_{i=2}^{m} P(\beta_i \mid \alpha_i)$.
```

Fig. 4. The original MO algorithm for estimating the probability $Q(s \mid \theta_D)$ [12]

where $\gamma > 0$ is a threshold parameter, which will be given in the following section. This gives the algorithm DCE1 in Fig. 3.

3.2 Probability Estimation for Strings by MO Method

To model $Q(d \mid \theta_D)$ for a sample D, we use the *MO method* (Maximal Overlap) method, introduced by Jagadish *et al.* [12] and built on top of an index structure based on the suffix tree [10].

The MO method 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. The *conditional probability* for a string α, given string β, is given by $P(\beta \mid \alpha) = P(\alpha\beta)/P(\beta)$. Let $s \in \Sigma^*$ be a *target* string to estimate $Q(s \mid \theta_D)$. In general, for any model $Q(\cdot \mid \theta)$, we can write the probability of the string $s = \beta_1 \cdots \beta_m$ ($m \geq 1$) where $s = \beta_1, \ldots, \beta_m \in \Sigma^*$, as $P(s) = P(\beta_1) \prod_{i=1}^{m} P(\beta_i \mid \beta_1 \cdots \beta_{i-1})$ For each $i = 1, \ldots, m$, the MO method approximates the conditional probability $P(\beta_i \mid \beta_{i-1} \cdots \beta_1)$ 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 strings that appears as substrings in D. In Fig. 4, we show the original MO method that computes an estimation of $P(s)$ in the greedy way. For every $i = 1, \ldots, |s|$ ($|s| \geq 1$), MO finds maximally overlapping substrings $\gamma_i = \alpha_i\beta_i \in Sub(\{s\})$ such that $\gamma_i \in Sub(D)$ and $\beta_i \neq \varepsilon$. Then, we define the associated probabilities by $P(\beta_i \mid \alpha_i) = fr(\beta_i\alpha_i)/fr(\beta_i)$. If there exists no such a 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 $s = \beta_1 \cdots \beta_{|s|}$. Finally, we compute the estimated probability by $Q(s \mid \theta_D) = P(\beta_1) \prod_{i=2}^{|s|} P(\beta_i \mid \alpha_i)$.

Jagadish *et al.* [12] 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 [13], which uses the non-overlapping decomposition of the input string in estimation.

3.3 Efficient Computation of MO by a Suffix Tree

In Fig. 6, we show the outline of algorithm DCE2 that runs in $O(N)$ to detect all spam candidates based on DCE, where $N = ||D||$ is the total length of documents. The crucial part of the original algorithm DCE1 in Fig. 3 is the suffix tree construction phase $\theta_{D \backslash \{d\}}$ at Line 4 and the MO-score estimation phase for $-\log Q(d \mid \theta_{D \backslash \{d\}})$ at Line 5 for each value of $d \in D$. This is a time-comsuming process since it iteratively builds the suffix tree $ST(D \backslash \{d\})$ M times for all except the current document. A straightforward implementation of DCE1 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. Fig. 5 shows a linear-time algorithm LinearMO for estimating $Q(s|\theta_D)$ using $ST(D)$ that quickly find the longest context β_i. The algorithm quickly finds the longest context α_i for each factor β_i by traversing the suffix tree using the suffix links via a technique similar to [10]. By the next lemma, LinearMO improves computation time of Line 4 of DCE1 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 Fig 5 correctly implements the algorithm* MO *in Fig 4 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 $\ell = |s|$ is the length of the string s.*

Proof. Let $\gamma_i = \alpha_i\beta_i$ $(m \geq 1, 1 \leq i \leq m)$ be the maximally overlapping substrings computed in the original MO. Assume that $\beta_j = x_h \cdots x_i$ $(h \leq i)$ with $k = |\beta_i|$. Then, we can show that $P(\beta_i \mid \alpha_i) = P(x_h \mid \alpha_i)P(x_{h+1} \mid \alpha_i x_h) \cdots P(x_i \mid \alpha_i x_h \cdots x_{i-1})$ since the longest context of x_j in s relative to $ST(D)$ is $\alpha x_h \cdots x_{j-1}$ for each $h \leq j \leq i$. Therefore, the correctness follows. The time complexity is derived from arguments similar to [10]. □

Speed-up of suffix tree construction. In the building phase at Line 3 of Algorithm DCE1, 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 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 $E' = \{ ([\alpha], [\beta]) \in V' \times V' : ([\alpha], [\beta]) \in E_D \}$. We define the frequency of a vertex $[\alpha]$ by $fr'([\alpha]) = fr_D([\alpha]) - fr_d([\alpha])$.

```
Algorithm LinearMO:
Input a string $s$, the suffix tree $ST(D)$for a set of strings $D$.
Onput an estimated probability of $Q(s|\theta_D)$.
 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 $(v, w)$ labeled 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 $Q(s|\theta_D)$ */
```

Fig. 5. A linear time MO-score estimation algorithm

Lemma 2. *Let $D \underset{\sim}{\subseteq} \Sigma^*$ 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 $\underset{\sim}{\ell,}$ we can simulate the algorithm* LinearMO *in Fig 5 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 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 Lema 1 and Lemma 3, we show the main theorem of this paper.

Theorem 1. *The algorithm* DCE2 *in Fig 6 runs in $O(N)$ time and $O(N)$ space, where $N = ||D||$ is the total size of D.*

3.4 Determining the Threshold

Our unsupervised spam detection method DCE2 in Fig. 6 takes a decision threshold $\gamma > 0$ as a learning parameter. To determine an appropriate value of γ, we first draw the histogram of the normalized document complexity L over unlabeled 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).

```
Algorithm DCE2(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$(d, ST(D \backslash \{d\}))$ 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. 6. An outline of our spam detection algorithm

Fig. 7. Histogram of the document complexity of Web data.

To justify this selection of γ, we show in Fig. 7 the histograms of the normalized document complexity L over all documents in a test collection YF (described later). In this experiment, documents are manually classified into three classes, the spam, the normal, and the whole document set to see the validity of the above selection method. In the figure, we see that the distribution for spam documents (spams) is concentrated at the values $L = 0.0 \sim 0.2$ (bit) per letter, while the normal documents (hams) spread at $L = 1.0 \sim 3.5$ (bit) per letter 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.

4 Experimental Results

4.1 Dataset

We used a test collection of forums from Yahoo Japan Finance[3], collected by Narisawa *et al.* [4]. The test collection consists of four sections of forum data: YF4314, YF4974, YF6830, and YF8473. All posts from each forum's data are labeled if they are spam. Table 1 shows the details of them. In the following experiments, we used only the body texts of documents, including the HTML tags.

[3] http://quote.yahoo.co.jp

Table 1. Details of the datasetsD

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

Table 2. Performance of spam detection methods.

Forum	Method	Recall	Precision	F-score
4314	DCE	0.59	**0.95**	**0.73**
	AM_Len	0.62	0.72	0.67
	AM_Siz	**0.82**	0.50	0.62
	AM_Max	0.68	0.62	0.65
	Naive	0.54	0.94	0.68
4974	DCE	**0.63**	0.69	**0.66**
	AM_Len	0.35	0.70	0.47
	AM_Siz	**0.63**	0.69	**0.66**
	AM_Max	0.57	**0.71**	0.63
	Naive	0.52	0.66	0.58

Forum	Method	Recall	Precision	F-score
6830	DCE	**0.68**	0.67	**0.67**
	AM_Len	0.41	0.66	0.51
	AM_Siz	0.75	0.58	0.65
	AM_Max	0.67	0.67	**0.67**
	Naive	0.54	**0.71**	0.62
8473	DCE	0.63	0.69	**0.66**
	AM_Len	0.53	0.72	0.61
	AM_Siz	**0.70**	0.61	0.65
	AM_Max	0.67	0.66	**0.66**
	Naive	0.54	**0.79**	0.64

4.2 Method

We implemented the proposed method 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.* [4] 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, called *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: Basic Performance. Table 2 shows the Recalls, the Precisions, and the F-scores for all methods. As shown in the table, DCE and AM_Siz achieved 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. 2: 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 Fig. 8 shows the Recall and the Precision curves of the dataset YF6830. To the left of the intersection of the Precision and the Recall

Fig. 8. Recall and Precision curve.

Fig. 9. Recall vs. density.

of DCE, Precision and Recall are clearly higher than for any other methods. However, to the right of the intersection, Precision decreases faster than for the other methods.

Exp. 3: Performance Against Noise Density. We created *noise spam* by using a seed as follows: For each position $1 \leq i \leq m$ of d, we replaced $d[i]$ by $d[j]$ with probability p, where $1 \leq j \leq m$ is a random number. We called the probability the *density* of noise. To examine the effect of the density, we set the number of spams to 20. Figure 9 shows Recall rates against density. Although DCE has higher Recall than Naive, its performance is insufficient at a higher density.

Exp. 4: Performance Against the Number of Spam Documents. In Fig. 10, we show the performance against the number of inserted spams. In this experiment, we used the density $p = 0.02$. According to the increase in the number of inserted spams, the performance of DCE showed improvement. However, Naive did not improve.

Exp. 5: Detecting Word Salads. *Word salad* is a new type of spam that is difficult to detect. Spammers create word salads by replacing words in a *skeleton* document with some interesting *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 and selected a spam as the skeleton in YF8473. The keyword set consisted of 15 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. AM_Siz detected about 20% of word salads despite only a few word salads being inserted. However, the Recall increased up to a final figure of only 30%. On the other hand, DCE detected approximately 100% of word salads when the number of word salads inserted was greater than 30.

5 Conclusion

In this paper, we have proposed an unsupervised spam detection algorithm that calculates document complexity. The algorithm runs in linear time w.r.t. the

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

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

total length of the input documents. We also conducted experiments using both real and synthetic data, where the real data was collected from popular bulletin boards and the synthetic data was generated as word salad spam documents.

;

References

1. Kolari, P., Java, A., Finin, T.: Characterizing the splogosphere. In: Proc. WWE'06, 2006.
2. Graham, P.: A Plan for Spam. `http://paulgraham.com/spam.html`, 2002.
3. Narisawa, K., Yamada, Y., Ikeda, D., Takeda, M.: Detecting Blog Spams using the Vocabulary Size of All Substrings in Their Copies. In: Proc. WWE'06, 2006.
4. Narisawa, K., Bannai, H., Hatano, K., Takeda, M.: Unsupervised spam detection based on string alienness measures. In Proc. DS'07, 161–172, LNAI, 2007.
5. Yoshida, K., Adachi, F., Washio, T., Motoda, H., Homma, T., Nakashima, A., Fujikawa, H., Yamazaki, K.: Density-based spam detector. In Proc. ACM KDD'04, 486–493, 2004.
6. Mishne, G., Carmel, D., Lempel, R.: Blocking Blog Spam with Language Model Disagreement. In Proc. AIRWeb'05, 2005.
7. Gyöngyi, Z., Garcia-Molina, H., Pedersen, J.: Combating Web Spam with TrustRank. In Proc. VLDB'04, 576–587, 2004.
8. Benczúr, A.A., Csalogány, K., Tamás Sarlós, M.U.: SpamRank – Fully Automatic Link Spam Detection. In Proc. AIRWeb'05, 2005.
9. Bratko, A., Filipič, B., Cormack, G.V., Lynam, T.R., Zupan, B.: Spam filtering using statistical data compression models. JMLR, **7** (2006) 2673–2698.
10. Ukkonen, E.: On-line construction of suffix-trees. Algorithmica, **14**(3) (1995) 249–260.
11. Cover, T.M., Thomas, J.A.: Elements of Information Theory. 2nd edn. Wiley (1938)
12. Jagadish, H.V., Ng, R.T., Srivastava, D.: Substring selectivity estimation. In Proc. PODS '99, 249–260, 1999.
13. Krishnan, P., Vitter, J.S., Iyer, B.: Estimating alphanumeric selectivity in the presence of wildcards. In Proc. SIGMOD'96, 282–293, 1996.