# 実システムへ適用可能な引用型スプログの検知手法の検証

## Identification of Plagiarism in Large Scale System

数見拓朗 [1,2*]　内藤遥 [1]
Takuro Kazumi[1,2], and Yo Naito[1]

[1] 株式会社サイバーエージェント
[1] CyberAgent, inc.
[2] 大阪大学大学院経済学研究科
[2] Graduate School of Economics, Osaka University

**Abstract:** Plagiarism, a kind of splog (spam blogs), is often found in blogs and can be harmful for maintaining blog services. For example, they have a bad influence on SEO and occupy a lot of space in storage. Therefore it is important for a company which operates a blog service to detect and remove. The purpose of our study is to identify a way to detect plagiarism quickly and with high precision. We use the near-duplicated detection algorithm suggested by Google to calculate the hamming distance, and then find any plagiarism in entries posted to Ameba Blog. We make two conclusions following our experiment. Firstly, we show that the algorithm is useful to detect plagiarism in blogs and is applicable to real systems. Secondly, using the algorithm, we can evaluate not only the hamming distance but also the jaccard index. In the experiment, we are able to detect plagiarism with high precision and almost constant time dependence on the size of the original data.

## 1　はじめに

近年，多くの人々が情報発信と情報収集を目的として，様々な場面でブログサービスを利用している．例えば，子供の成長を投稿するユーザや，お気に入りの芸能人の投稿を閲覧するユーザなどがいる．また，ある商品やサービスを利用した一部のユーザが，その評価をブログに書き込むことがあるので，企業のマーケティング担当者にとって，ブログは重要な情報源になっている．

一方で，ブログサイトには，悪意のあるユーザによってスパムブログ（スプログ）が大量に投稿されることがある．[6] によると，スプログは次のような６つのタイプに分類することができる．(1) ニュース記事などをコピーし生成した「引用型」，(2) アフィリエイトリンクのみで構成され，オリジナルな記述をほとんど含まない「アフィリエイト型」，(3) 卑猥な文書が書かれている「アダルト型」，(4) 複数の情報源から単語を抽出し，それらを並び替え生成した「ワードサラダ型」，(5) ギャンブルや，FX で設ける方法などの話題に閉じている「ギャンブル・金融型」，(6) 商品が列挙されたショッピングサイトである「ショッピング型」である．このようなスプログの増加は，ブログサービスを利用するユーザを不快にさせるだけではなく，ブログを使って何らかの分析を行なうユーザにも悪影響を与え，メディアとしての価値を大きく低下させる．

株式会社サイバーエージェント [1] が提供するブログサービスである，アメーバブログ [2]（アメブロ）でも，上記のようなスプログが少数ながら投稿されており，スプログへの対応は，サービスを運営する上で重要である [7]．サービス運営上，スプログ検出で課題となるのは，主に以下の２点である．第一に，識別性能である．通常の投稿をスプログと誤って判定し，ユーザに警告などを行なうと，ユーザに不信感を抱かせてしまう．一方，スプログを多く取りこぼしてしまうとアメブロのドメイン価値が低下し，検索流入が大きく減少してしまうかもしれない．このように，低い識別性能は，サービスに悪影響を与えるので，識別性能はできるだけ高く維持する必要がある．

第二に，スプログの検出速度である．日々大量のデータが投稿されるので，スプログ検出をオンライン，あるいはオフラインで行なう場合でも高速な処理が求められる．特に，アメブロでは，一日当たり，約数十万の記事が投稿されているので，データ量が増加したと

*連絡先：株式会社サイバーエージェント
東京都千代田区外神田 1-18-13 秋葉原ダイビル 8F
kazumi_takuro@cyberagent.co.jp

[1] https://www.cyberagent.co.jp/
[2] http://ameblo.jp/

しても検出速度を一定の水準に保つことは重要な課題である．

本稿の目的は，上記のような課題を克服して，実システムに適用可能なスプログ検出の手法を検証することである．本稿では，特にアメブロ内の「引用型」スプログに注目し，これを高速に検出する方法を示す．

検出対象とする「引用型」スプログは，次のような２種類に大別される．第一に，アメブロに既に投稿されている記事をコピーし，何度も投稿する「内部引用型」スプログである．これは，何らかの宣伝を行なっている記事に多く見受けられるタイプである．

第二に，ニュース記事など，アメブロ外で投稿されている記事をコピーして，投稿する「外部引用型」スプログである．このような「外部引用型」スプログは，アメブロ外のデータをコピーしているので，スプログを検出するには，外部のニュースサイトをクローリングして，蓄積しておく必要がある．

「内部引用型」スプログと「外部引用型」スプログに共通する特徴として，文章の前後に，数行だけ関係のない話題や感想，その日の天候などが付け加えられている．

貢献として以下の２点を強調したい．第一に，実システムに適用可能な設計を提案することである．本稿では,「引用型」スプログを検出するアルゴリズムとして [1] を用いて，アメブロに実際に投稿されたブログを利用し，その精度と検出速度を確認する．第ニに，[1] のアルゴリズムを b-bit minwise hashing [2] に応用して，Jaccard 係数の推定値を高速に求めることである．

本稿の構成は以下の通りである．第 2 章では，本稿と関連の深い先行研究について紹介をする．第 3 章では，本稿の実験に利用したデータについて説明をする．第 4 章では，実験の方法と結果について詳細に説明をする．最後に，第 5 章で本稿の結論を論じる．

## 2　関連研究

「引用型」スプログの判定には，何らかの類似度判定のアルゴリズムを利用することが多い [1, 3, 4, 6]．これは,「引用型」スプログのほとんどが，ニュース記事などのコピー元記事を少し改変したものを掲載しているからである．

本研究は，[1] と深く関連している．[1] の貢献は，Charikar の simhash [5] が，大規模なデータから重複しているコンテンツを抽出するに有効であることを示しており，ハミング距離が閾値以下のデータを高速に取得する方法を提案していることである．さらに，64 ビット長のハッシュ値を採用する場合，ハミング距離が 3 以下になるペアを類似していると見なすことが，F 値が最も良くなることを明らかにしている．

[1] と同様，類似している記事の発見に simhash を用いる研究として，[3] が挙げられる．[3] では，文書のクラスタリングを行い，類似しているコンテンツを効率的に抽出できることを示している．判定前に記事のクラスタリングを行なうことで，対象となる記事を絞り込み，データ量の増加にも対応できることを示している．

[4] は，携帯メールなどのショートメッセージの重複コンテンツを抽出する際にも，simhash アルゴリズムが有効であることを示している．[1] と異なる点は，50 文字以内の短い文章を対象としたとき，類似判定の基準をハミング距離 3 以下とすることが必ずしもよいパラメータとはなっていないということである．

上記の研究では，全て日本語以外で書かれた文章を対象にしているが，[6] は，日本語で書かれたスプログの調査と「引用型」スプログの判定手法の提案をしている．彼らの研究の貢献は主に２つある．第一に，27 個の主要ブログサイトから記事を収集し，スプログを６種類に分類できることを明らかにしている．調査の結果,「引用型」スプログが，他のスプログと比較して，最も多く投稿されていると報告している．第二に，転置インデックスを応用した「引用型」スプログの判定手法について提案していることである．

## 3　データ

本稿では，実際にアメブロに投稿された記事に対して，過去に投稿されている記事の中からコピーされているかを判定する．判定の対象となる記事は，2015 年 10 月 6 日にアメブロに投稿された記事からランダムにサンプリングした 10 万件である．コピー元データとして利用するのは，2015 年 10 月 1 日から 2015 年 10 月 5 日にアメブロに投稿された記事を，一日ごとに 10 万件ずつランダムサンプリングした合計 50 万件である．

## 4　検知手法と結果

本章では，実験に利用する検出手法，実装の概要，実験結果についてそれぞれ報告する．

### 4.1　検出手法

ユーザが投稿した記事が「引用型」スプログかどうかを判別する流れを図 1 に沿って説明する．処理は，オンライン処理部分とバッチ処理部分の２つに分かれている．オンライン処理部分は，投稿内容をハッシュ値に変換する前処理を行ってから，データベースに問い合わせ，ユーザ投稿が「引用型」スプログかそうでないか

を判別する処理を行う．バッチ処理部分では，ニュース記事などコピー元となる記事をハッシュ値に変換する前処理を行い，データベースに格納する処理を行なっている．尚，オンライン処理とバッチ処理のデータの前処理は同様の処理を行なっている．

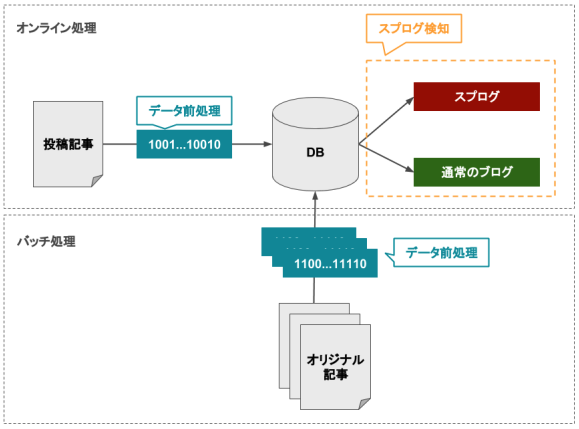


図 1: 処理の流れ

オンライン処理部分で,「引用型」スプログを抽出する際に，２つの類似度指標を用いる．第一に，ハミング距離である．ハミング距離は，値域が 0 以上の整数値であり，２つのニ値ベクトルの論理的排他和から導出することができる．例えば，$x = 11010, y = 10101$ である 2 つのハッシュ値のハミング距離は 4 である．ハミング距離が小さければ，２つのベクトルは類似していると言える．実験では，ハミング距離が閾値 $h$ 以下のものを「引用型」スプログとして抽出し，閾値ごとの精度と，データサイズごとの検出速度を報告する．

第二に，Jaccard 係数である．Jaccard 係数は，２つの集合の類似度を表し，0 から 1 の値をとる．0 に近づけば２つの集合は類似していないと判定でき，1 に近づけば，類似していると見なすことができる．たとえば，集合の要素数を次元とするニ値ベクトル $x = 11010, y = 10101$ の Jaccard 係数は，0.2 である．次元数が小さいとき，Jaccard 係数の計算時間は問題にはならないが，次元数が大きくなると Jaccard 係数の計算時間が長くなる．そこで，b-bit minwise hashing[2] を用いて Jaccard 係数の近似値 $\tilde{J}$ を求めることにする．ハミング距離と同様に，$\tilde{J}$ が閾値 $j$ 以上のものを「引用型」スプログとして抽出する．

b-bit minwise hashing では，以下のような手順で，$\tilde{J}$ を求めることができる．まず，異なる $k$ 個のハッシュ関数 $f_1, f_2, ..., f_k$ を用意する．次に，集合 $S$ の全要素にハッシュ関数 $f_i$ を適用して，ハッシュ値を得る．得られたハッシュ値の中から最小値を取り出す．最後に，取り出した最小値の下位 $b$ ビットを保存する．これをハッシュ関数の個数分だけ繰り返し，$kb$ ビット長のハッシュ値を求める．以上の手順で求めたハッシュ値を $bBit(S)$ とする．[2] では，$\tilde{J}$ は以下のような式で導出可能であることを証明している．

$$\tilde{J} = \frac{\frac{k-|bBit(S_1) \oplus bBit(S_2)|}{k} - 2^{-b}}{1 - 2^{-b}} \quad (1)$$

実験結果では，ハミング距離と同様に，閾値ごとの精度と，データサイズごとの検出速度を報告する．尚，本稿では $b = 1$ として実験を行なう．

### 4.1.1 前処理

ここでは，記事をハッシュ値に変換する前処理について説明する．ハッシュ値の変換には，Chariker の提案する simhash と b-bit minwise hashing の 2 つの手法を用いる．

**Step 1**
投稿された記事の文字数をカウントし，50 文字を超えるものを取得する．

**Step 2**
投稿された記事から文字 bi-gram を得る．

**Step 3**
Step 2 で作成した文字 bi-gram を simhash，あるいは b-bit minwise hashing を利用してハッシュ値に変換する．

上記のように作成したハッシュ値をデータベースへ格納する．これを次節 4.1.2 で説明する．

### 4.1.2 データベースへの格納

次に，4.1.1 の前処理によって作成したハッシュ値をデータベースへ格納する方法について説明する．

**Step 1**
ハッシュ値を格納するために，$t$ 個のテーブル $T$ を用意する．

**Step 2**
あらかじめハッシュ値を分割し組み合わせるルールをテーブルごとに決めておき，そのルールに従って新たな $t$ 個のハッシュ値を作成し，各テーブルに格納する．

**Step 3**
テーブル内のハッシュ値を昇順にソートする．

以上が，図 1 のバッチ処理に該当する処理である．バッチ処理部分では，上記の処理を定期的に実行し，コピー元記事をデータベースに保存している．次節 4.1.3 では，投稿記事が「引用型」スプログかどうかを判定する，オンライン処理部分を説明する．

#### 4.1.3 「引用型」スプログの検出

最後に，投稿された記事 $Q$ が,「引用型」スプログかどうかの判定を行なう処理を説明する．これは，図 1 のオンライン処理にあたる．

4.1.1 の前処理によって作成したハッシュ値を，4.1.2 の Step 2 で定めたテーブルごとのルールに従って，ハッシュ値 $q_i$ を作成する．これと，先頭の $p_i$ ビットが一致しているものをテーブル $T_i$ から抽出する．この中からハミング距離，あるいは Jaccard 係数の推定値を計算する．これをテーブル数 $t$ だけ繰り返し，ハミング距離の閾値 $h$ 以下，あるいは Jaccard 係数の閾値 $j$ 以上を「引用型」スプログとする．

### 4.2 実装

実験では，データベースに，カラム指向型データベースの一種である HBase を利用する．HBase では，RowKey と Value を使ってキーバリューストアとしてデータを扱うことができる．また各 Value ではタイムスタンプを使って複数のバージョンを保存することができる．RowKey には，テーブル番号 $i$ と，並び替えたハッシュ値の先頭 $p$ ビットを結合したものを登録する．

一方，Value には，記事の属性情報とハッシュ値を結合したものを登録する.「引用型」スプログを検索する際には,「引用型」スプログかどうかを確認したい投稿をハッシュ値に変換し並び替え，その先頭 $p$ ビットとテーブル番号 $i$ を組み合わせて RowKey とする．RowKey の数だけ並列に HBase への問い合わせを行い，Value を取得する [3].

### 4.3 結果

本節では,「引用型」スプログの検出精度と速度についてそれぞれ報告する．

#### 4.3.1 精度

「引用型」スプログの検出精度をハミング距離と Jaccard 係数に分けて，それぞれ報告する．ここで精度とは,「引用型」スプログとして検出された投稿の中で，人手でスプログと判断された投稿の割合とする．具体的には，10 万件のうち,「引用型」スプログとして検出されたものを人手で判断する．ただし，人手で判断するのは，ランダムに選んだ 50 件とする．

まず，ハミング距離と精度の関係を図 2 に示す．それぞれの点は，ハミング距離 $h \in \{1, 2, 3, 4, 5, 6\}$ のときの精度を表している．ハミング距離を大きく取れば「引用型」スプログではない投稿を多く検出する傾向がある．これはハミング距離が非類似度を表しており，$h$ を大きく取ると類似していない投稿を多く含むようになるからである．

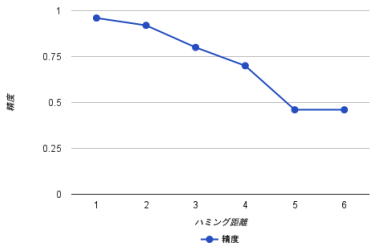


図 2: ハミング距離と精度

次に，Jaccard 係数と精度の関係を図 3 に示す．それぞれの点は，Jaccard 係数 $j \in \{0.7, 0.8, 0.9, 1.0\}$ のときの精度を表している．図 3 が示す通り，Jaccard 係数が大きくなるにつれて，検知するデータの中で「引用型」スプログの割合が多くなり，精度が上昇する．

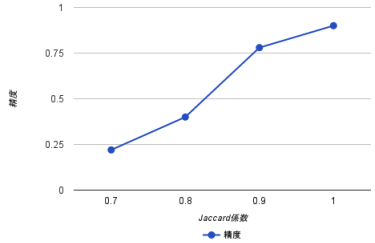


図 3: Jaccard 係数と精度

ハミング距離と Jaccard 係数のいずれの場合でも，誤って検出したブログは次のような 2 種類の分類される．第一に，Instagram[4] やパシャっと my ペット [5] からアメブロに投稿された，写真と 50 文字以上の写真のタイトルや感想を組み合わせた投稿である．全く異なる画像ではあるが，感想文などが似ているために「引用型」スプログとして検出してしまうケースが観察された．これを解決するには，画像認識の判別を行なう必要がある．

第二に，スクリプトを埋め込んだ投稿である．投稿をブラウザが読み込んだタイミングで文章が展開されるため，今回の実験では本文に記載されていた URL が似ていたため誤って検出された．スクリプトが埋め込まれている投稿は,「引用型」スプログではないタイプのスプログである可能性が高い．今回，本稿で検証したアルゴリズムとは異なるアルゴリズムで検出する必要がある．

[3] 実験では，HBase の MultiGet 機能を利用している．

[4] https://instagram.com/

[5] http://petpic.jp/

#### 4.3.2　検出速度

コピー元となる記事の量と「引用型」スプログの検出速度の関係を，ハミング距離と Jaccard 係数のそれぞれについて明らかにする．

検出速度の実験の概要は以下の通りである．まず，コピー元記事の 10 万件を HBase に格納する．テストデータである 10 万件のデータを全て判定するのにかかる時間を 5 回計測し，平均を求める．その後，コピー元記事を 10 万件ずつ 50 万件まで増やし，同様の実験を行なう．

図 4 は，ハミング距離を求めるときの，データサイズと検出速度の関係を表している．凡例は，ハミング距離を示している．それぞれの点は，コピー元記事が 10 万件，20 万件，30 万件，40 万件，50 万件のいずれかが与えられたときに，10 万件の投稿を全て判定するのにかかる平均検出時間に対応している．検出速度が増加する要因を 2 つに分けて説明する．第一に，ハミング距離が大きくなるにつれて検出時間が増加しているのは，テーブル数 $t$ が増えて HBase への問い合わせ回数が多くなるからである．第二に，ハミング距離が大きい場合，データサイズに対して線形に検出時間が増加するのは，HBase から取得するコピー元記事の数が増え，判定に時間がかかるためである．

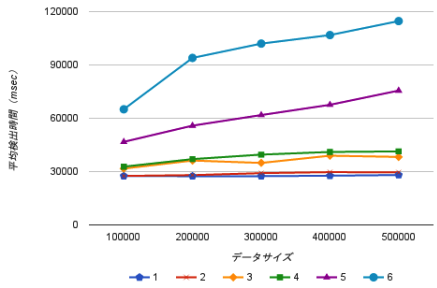


図 4: ハミング距離の検出速度

図 5 は，Jaccard 係数の推定値が 0.9 以上の記事を取得するときの，データサイズと検出速度の関係を表している．凡例は Jaccard 係数を推定するのに必要とするハッシュ値のビット長を示している．ハミング距離と同様に，データサイズによらず検出速度はほぼ一定であるが，ハッシュ値のサイズを大きくすると，テーブル数 $t$ が増えて HBase に問い合わせる数が増加するので，検出速度は遅くなる．

## 5　まとめ

本稿では，アメブロに投稿される「引用型」スプログの高速な検出手法について検証を行なった．実験結果から，スプログを高い精度で，かつデータサイズに依存せず高速に検出できることを示した．また，[1] のアルゴリズムを応用して，ハミング距離だけでなく，Jaccard 係数の推定値の導出を行なった．

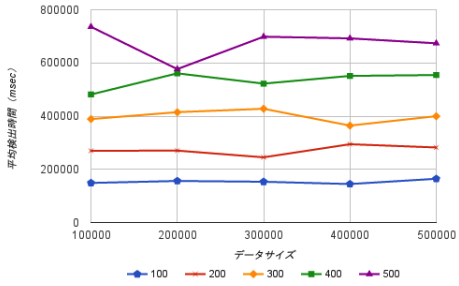


図 5: Jaccard 係数の検出速度

今後の課題は，以下の 3 点である．第一に,「引用型」スプログの判定を，文字ベースだけではなく，画像認識と組み合わせることである．第二に，投稿にスクリプトが埋め込まれている場合の対応である．第三に，再現率を計測することである．本研究では，精度のみの検証を行なっているが,「引用型」スプログの判定基準を変化させたときに，どれだけ取りこぼしなく「引用型」スプログを抽出できたか不明である．今後はこの 3 つに留意して拡張していく．

## 謝辞

本稿の作成にあたり，福田一郎氏（株式会社サイバーエージェント）から数多くの研究支援を頂いた．もちろん，本稿中のすべての誤りは筆者に期するものである．

## 参考文献

[1] Manku, Gurmeet Singh, Arvind Jain, and Anish Das Sarma. ”Detecting near-duplicates for web crawling.” Proceedings of the 16th international conference on World Wide Web. ACM, 2007.

[2] Li, Ping, and Christian König. ”b-Bit minwise hashing.” Proceedings of the 19th international conference on World wide web. ACM, 2010.

[3] Kumar, J. Prasanna, and P. Govindarajulu. ”Near-duplicate web page detection: an efficient approach using clustering, sentence feature and fingerprinting.” International Journal of Computational Intelligence Systems 6.1. pp 1-13. 2013.

[4] Pi, Bingfeng, et al. ”SimHash-based Effective and Efficient Detecting of Near-Duplicate Short Messages.” Proceedings of the 2nd Symposium International Computer Science and Computational Technology. 2009.

[5] Charikar, Moses S. "Similarity estimation techniques from rounding algorithms." Proceedings of the thity-fourth annual ACM symposium on Theory of computing. ACM, 2002.

[6] 高橋 哲朗, 野田 雄也, 岩倉友哉: スプログの調査と実システムにおける判別手法, 第 15 回言語処理学会大会, 2009.

[7] 數見拓朗: アメーバブログにおけるスパムブログ検知: 機械学習を用いたスパムフィルタの開発, 第 7 回 Web とデータベースに関するフォーラム, 2014.