2015年度 卒業論文

ソーシャルメディア利用環境の改善提案

大阪産業大学 デザイン工学部 情報システム学科
情報教育システム研究室

12H068 田崎慧

目次

1　はじめに

近年、スマートフォンやタブレット端末の普及によって一般層がインターネットに触れる機会が大幅に増加した。インターネットの利用目的の中でも、若年層では SNS(ソーシャル・ネットワーキング・サービス) が大部分を占めている [1,2]。個人が自分の主張したい考え・意見を、ウェブログのような日記感覚ではなくリアルタイム性に優れたツールで発信できるようになっている。その手軽さが人気の理由である。しかし、発信する手軽さ故のトラブルがある。発信内容によっては、その情報を受け取る人物が誤解や反感を持つ、レッテルを貼られるなどのコミュニケーション上の問題 (コミュニケーションギャップ) が起こる場合がある。そこから広く拡散され、多方向から多数の批判コメントを受ける「炎上」と称される現象が起きる。炎上に代表されるソーシャルメディア上のトラブルは、社会問題として大きく注目されている [3]。

第 2 章では本研究の目的について述べる。第 3 章ではソーシャルメディアの現状について述べる。第 4 章では製作したシステムについて説明し、第 5 章ではシステムの動作結果をその考察とともに述べる。第 6 章では今後の課題について述べる。第 7 章では研究の成果についてまとめる。

2 本研究の目的

本研究の目的は、ソーシャルメディアを利用する上で情報を受け取る者と発信する者の間に起こる「意図しない形」での人間関係の衝突を防ぐことである。ユーザのタイムライン[*1]に流れるツイート[*2]に対する一操作として、ユーザの好まないであろう要素が含まれているものにマスクをかける。それにより、ユーザがコミュニケーションを円滑に行う支援ができ、なおかつユーザが炎上することなくサービスの利用を継続できるようになる。

現状、見たくないツイートに対しての対処法はミュート機能・ブロック機能・スパム報告機能があるが、それらはユーザそのもの、もしくは単純なキーワードマッチングのみで非表示にしており、本来ユーザが見たい情報まで隠してしまう。

本研究ではユーザの意向に沿った表示が成せるよう、スパムメールの判定に用いられるベイジアンフィルタを使う。ツイート内容からユーザが好まないであろう要素を特定し、該当しているであろうツイートに対して非表示用のマスクを被せた。

[*1] 自分がフォローしているユーザの投稿した内容が時系列に並ぶログのこと。

[*2] twitter における投稿のこと。投稿内容は 140 文字以内という制限がある。

3 ソーシャルメディアの利用環境

3.1 ソーシャルメディアとは

ソーシャルメディアとは、インターネット上で個人や団体が情報を発信したり、双方向でコミュニケーションを取ることが可能なメディアのことである。電子掲示板やレビューサイト、ウェブログ、画像・動画共有サイト、ソーシャル・ネットワーキング・サービス (SNS) などがソーシャルメディアの例として挙げられる。中でも Facebook や Twitter, LINE は特に人気を博している。リアルタイム性・拡散性・対話性といった、今までのインターネットメディアとは異なる性質を持ったことにより、一般層にも広く受け入れられたと考えられる。また、スマートフォンやタブレット端末といったデバイスが急速に普及し、アプリケーションという形で利用しやすくなったことも人気の理由である。

3.2 炎上

ソーシャルメディアに関連する、社会的な問題として「炎上」が注目された。炎上したかどうかを見極める明確な定義は定まっていないものの、ユーザの想定を超え、誹謗中傷や批判などのコメント・トラックバックが殺到すること[*3]が炎上であるという見方が主である。

ソーシャルメディアの中で炎上が起こる経緯は以下の通りである。

1. あるユーザが「問題のある内容」を投稿する
2. 投稿を受け取った他ユーザがそれに反応、拡散し、発信ユーザに対してコメントなどのアクションを起こす
3. 拡散された投稿が更に拡散され、他ユーザが同調・便乗、同じくコメントなどが増える
4. ネットニュースやまとめサイト[*4]などに取り上げられる

上記した経緯の他には、著名人の TV 番組内やライブストリーミング配信での発言など既存のメディアから「飛び火」するパターンなどが存在する。企業の場合は顧客に対しての対応が、顧客側から不満の出るものであった場合に拡散される可能性がある。「問題のある内容」には、反社会的行動の自慢・著名人や社会的弱者への暴言 (悪口) といったものから、政治思想や宗教、ジェンダー、社会問題関連の話題といった個人の主義主張の違い、また人によって受け取り方の異なる話題が該当する。そういった事象に至るのは、匿名も可という発言のしやすさ、内輪と公共との区別がついていない、情報のネットリテラシーの欠如などが原因となっている。内輪と公共の区別、受け取り方の違いという面においては、発信者は炎上を意図せず普段通り発言しているが、第三者が文脈を無視し事実だけを受け取っているという見方が存在する [4]。人毎に異なる規範が存在し、炎上を狙ったものでない発言に対しても、憤りや道義心を感じるユーザは存在する。これは、ソーシャルメディアを利用しているユーザの繋がりが広いほどに起こり得る。

3.3 炎上対策

社会問題として大きくメディアに取り上げられた炎上だが、予防対策法が考えられている。前項にて記述したような「問題のある内容」を投稿しないことは基本である。極力炎上しそうであると考えられる話題を投稿前に学んでおくことで、炎上するリスクを抑えることが出来る。また、炎上そのものを特定する研究 [5] などが行われており、世評を分析し炎上事例を収集、未来の炎上予測を行うことができている。予測によって投稿する文章を推敲する機会を生み、炎上を避けるような利用が可能になるとしている。炎上の可能性がある投稿を防止するにあた

[*3] ユーザが企図しているものは「釣り」と呼ばれる。「炎上商法」というマーケティング法も同じ。

[*4] 特定の話題について情報をまとめたサイト。

り、システム側で文章そのものを変えてしまうことは、同一性保持権を侵害する場合がある。情報の発信者側から炎上を防止する際には、ユーザの権利を侵害しないような配慮が必要である。

他人を攻撃し、炎上させることを目的としている人物の存在について触れている意見もある [6]。私怨による他者への誹謗中傷や、面白半分での攻撃、特定対象を攻撃している他者への同調などが理由として考えられる。そういった場合には、対処するだけ負荷がかかるのみなので、自分への攻撃をスルーするスキルや毅然と構える態度が必要であるとしている。

4 Web アプリケーション

4.1 システムの概要

本システムは、ツイートをフィルタリングして、引っかかったものにマスクを掛けて隠す Web アプリケーションである。これを KLIENT と命名した。必要な実装箇所を挙げていくと、以下の実装が必要となる。

- ユーザのタイムラインを取得し、ツイートを表示する
- 好まないツイートを分類するボタンを動作させる
- ツイート本文を単語毎に分解する
- 分解された単語をスパムなもの/スパムでないものに分類する
- 分類された単語と出現回数を保管する
- 保管された単語と出現回数から分類器を生成する
- 分類器を使い、ツイート本文をスパムかスパムでないか判定する
- スパムと判定された場合に該当ツイートをマスクを掛けて隠す
- マスクをクリックすると隠されたツイート本文を表示する

今回の実装では localhost 上で Web サーバを立ち上げ、ツイート内容を解析するベイジアンフィルタを動作させている。Web ブラウザ上でユーザ毎のタイムラインが表示される Twitter クライアントを製作し、フィルタによりスパムであると判断されたツイートに対してマスクを掛ける。

4.2 システムの開発・動作環境

制作したシステムの開発・動作環境について説明する。

- 開発・動作環境
  - 端末 …… Macbook Pro
  - OS …… OS X (10.9.5)
  - データベース …… SQLite3 (3.7.13)
  - 言語 …… Ruby (2.0.0p645)
  - ブラウザ …… chromium, firefox
- 必要な RubyGems[*5]
  - sinatra
  - sinatra/reloader
  - twitter
  - haml
  - sqlite3
  - active_record
  - sinatra/activerecord
  - json
  - omniauth
  - omniauth-twitter
  - dotenv

[*5] Ruby 言語用のパッケージ管理システム。ライブラリは「gem」と呼ばれる。

– addressable/uri
– mecab
– classifier

4.3 システムの内容

本研究はユーザのタイムラインに流れるツイートをフィルタリングし、マスクを掛けるプログラムである (実際の動作画面を図 1, 図 2 へ示す)。本システムでは、Sinatra という Web フレームワークを用いて Web アプリを動作させている [7]。実際にユーザのタイムラインを表示させる部分は、Haml テンプレートエンジンを用いて HTML を出力している [8]。HTML のタイムラインに表示されるツイートは Twitter REST API [9] を利用し取得している。API の仕様上、15 分に 15 回のみツイートを取得できるように制限されている。流れてくるツイートをフィルタリングする部分 (=スパムフィルタ) に関しては、ベイジアンフィルタ [10] を利用した。また、判定基準となる要素を SQLite に構築した DB へ保存するため、ツイート本文の横にスパム報告・誤判定報告の入力ボタンを設置している (設置した入力ボタンを図 3, 図 4 に指す)。これを押して入力を行ってもらうことにより、ユーザの意向に沿った形でのフィルタリングを目指した。フィルタリング対象と判断されたツイートに対しては、本文にマスクを掛けて表示している。マスクを掛けた部分を図 5 に示す。本文データの受け渡しに関しては、Ajax を利用している。

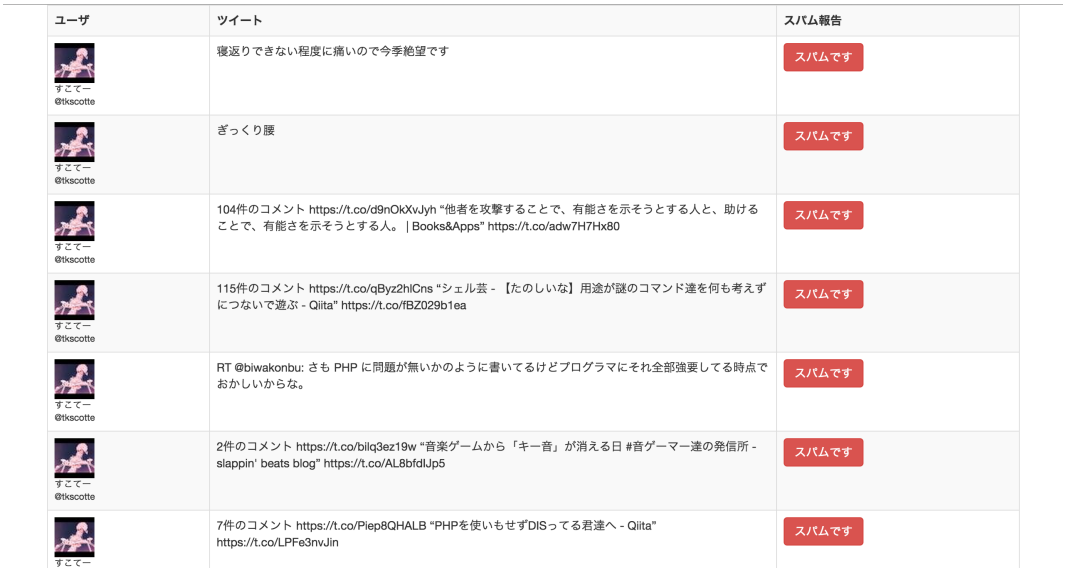

図 1　実際の動作画面。一般的なクライアントにあるような、ユーザのタイムラインに流れてくるツイートを表示している図。

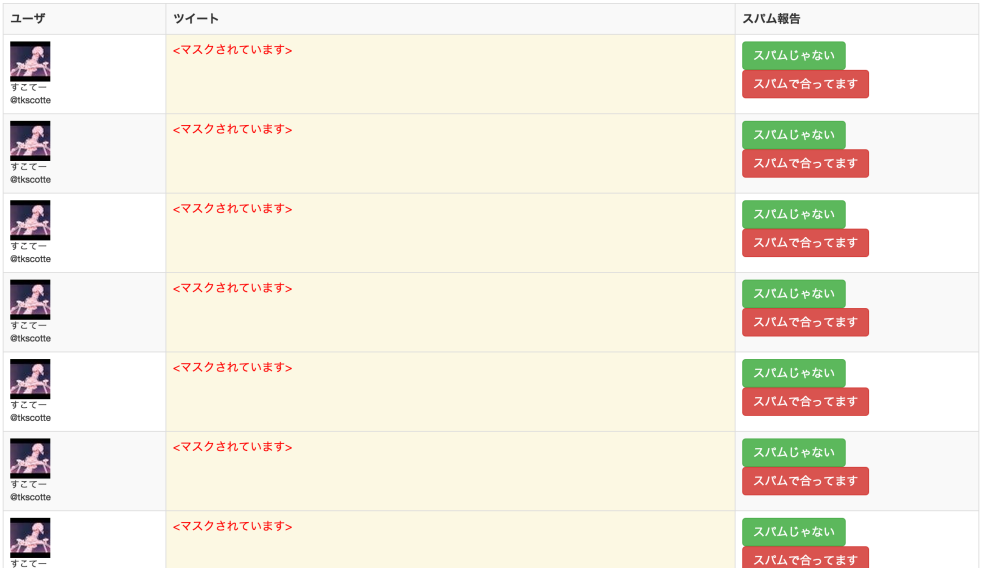

図 2　実際の動作画面。本研究ではこの図のように、ユーザが不快に感じると考えられるツイートに対し「マスク」を掛けている。

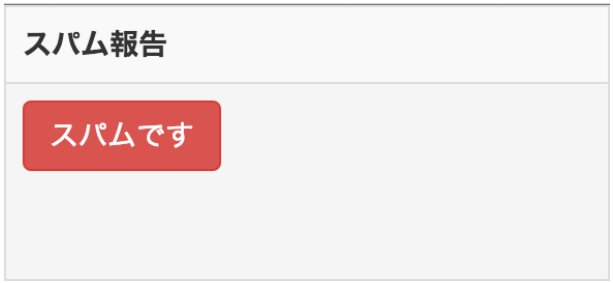

図 3　スパム報告ボタン。不快なツイートと感じた場合には「スパムです」ボタンをクリックする。それにより、スパム用テーブルへ各要素を登録している。

図 4　スパム報告/誤判定報告ボタン。マスクされているツイート本文を実際に見て、不快でないと感じた場合には「スパムじゃない」ボタンをクリックする。逆に不快なものであると認識した場合には「スパムで合ってます」ボタンをクリックする。

<マスクされています>

DANCEのカタログ見たいよ〜〜早く公開してくれ〜〜

おそようございます

図 5　マスク部分の動作例。ツイートを取得した際に解析し、文章がスパムかスパムでないかの確率を算出。その結果からスパムと判断された場合に「マスクされています」というメッセージと共にツイート本文が非表示となる。マスクされたツイート本文を見たい場合にはマスク部分をクリックする。

4.4　ユーザ認証

多数の Twitter ユーザに利用してもらうためには、OAuth 認証などを設け、ユーザ毎の DB を構築する必要がある。本実装では OAuth 用のライブラリを利用し、認証画面を設けた。以下に認証画面を提示する。(図 6)

図 6　OAuth 認証の画面。ユーザがアプリケーションに対し、ユーザ情報を利用して良いか確認を行う。

4.5 スパムフィルタ

本研究を行うにあたり、今日メーラーソフトにて用いられるスパムフィルタを参考にした。スパムフィルタにも様々なフィルタリング手法が存在する [11]。以下にその例を挙げる。

キーワードフィルタ
: 有害と考えるキーワードやフレーズをあらかじめ登録しておき、表示したいものがマッチしている場合にフィルタリングする手法。

ベイジアンフィルタ
: 4.5.1 にて後述する。

ロジスティック回帰
: 過去の事例から統計的に解析する手法。複雑な条件を設定できる。

ハニーポット
: 囮となるユーザを置いておき (無人)、そのユーザに送られてきたものはスパムとする手法。最終的には人力で判定を行う。

ヒューリスティクスフィルタ
: スパムとする様々なルールをユーザや管理者が定義していき、それに引っ掛かったものをスパムにする手法。

また、現状 Twitter が提供している機能として以下の機能が存在する。

ミュート機能
: 特定のユーザが投稿したツイートを、自分のタイムラインで非表示にする機能。[12] ミュートされたユーザは、自分がミュートされているか気付かないようになっている。

ブロック機能
: 特定のユーザが自分をフォローしたり、自分の投稿したツイートを見たりできないようにする機能。[13] ブロックされたユーザには「ブロックされた」という通知が飛ばないが、ユーザのプロフィールを見に行くことで自身がブロックされていることがわかるようになっている。

スパム報告機能
: 特定のユーザが悪質なアカウントであると運営に報告し、ブロック機能と同じ処理を行う。[14]

上述したフィルタリング手法と照らし合わせると、ミュート機能・ブロック機能・スパム報告機能はキーワードフィルタとヒューリスティクスフィルタに相当する。本システムでは、それらにない形での利用環境改善を目指し、ベイジアンフィルタを利用している。

4.5.1 ベイジアンフィルタについて

今日メーラーで用いられるフィルタの一つに、ベイジアンフィルタが存在する。ベイジアンフィルタは、ポール・グラハム氏が 2002 年に発表した論文に由来するものである [15]。主に自然言語の文章の分類に用いられている。メーラーにおけるベイジアンフィルタの仕組みは以下の通りである。

1.「大量の迷惑メール」と「大量の通常メール」の単語の分布を算出する
2. 各単語が迷惑メール内、通常メール内で出現する確率を算出する
3. 各単語のスコアを求め、分類辞書へ記録していく
4. 分類辞書を利用し、メールの文章の判定用スコアを算出する判定用スコアを上回るものは迷惑メールと判定される下回るものは通常メールと判定される
5. フィルタの性能を上げるために判定用スコアを調整する

本研究では、ツイート本文が「スパムである」か「スパムでない」かを分類する。本実装でのベイジアンフィルタの動きは以下の通りである。

1. ツイート本文を単語に分割する
2. 分割した単語を分類辞書と照合する
3. マッチした単語から判定用のスコアを加算する
4. ツイート本文中の単語を全てチェックし、判定用のスコアを超えている場合に本文をスパムと分類する

スコアの計算はベイズの定理を元に行われる。入力 X が与えられた時、入力 Y が得られる確率は以下の数式で求められる。

$$P(Y|X) = \frac{P(Y)P(X|Y)}{P(X)} \tag{1}$$

上記がベイズの定理である。確認したい文章がカテゴリに入るかどうかを判定する場合には、D が文書 (Document)、C をカテゴリ (Category) とした場合に以下の数式で求められる。

$$P(D|C) = \frac{\text{文書 } D \text{ の数}}{\text{カテゴリ } C \text{ に属する文書数}} \tag{2}$$

となる。このままでは、文書全体が一致しているかどうかという判定になるため、ベイジアンフィルタでは文書を単語毎に分解し判断する。W が単語 (Word)、C がカテゴリとして以下の数式に表せる。

$$P(W_1, W_2, ..., W_n|C) = P(W_1|C) \times P(W_2|C) \times ... \times P(W_n|C) \tag{3}$$

上記がナイーブベイズである。ここから、スパムか否かの分類を行いたい場合などにはナイーブベイズ分類器というものを用いる [16]。システムに当て嵌めて数式を記述すると、

$$\text{単語のスコア} = \frac{\text{スパムテーブル内での出現確率}}{\text{スパムテーブル内での出現確率} + \text{非スパムテーブル内での出現確率}} \tag{4}$$

でスパムか非スパムかの判定を行うこととなる。

しかし、このままでは新しく出てきた単語を用いる際に値が 0 となる「ゼロ頻度問題」が起こる。そのため、加算スムージングと呼ばれる手法を使い、この問題を緩和する。本研究では加算スムージングの中でも、ラプラススムージングという手法を用いた。ラプラススムージングの場合、以下の数式に表せる。

$$P(W_i|C) = \frac{\text{単語 } W_i\text{の数}}{\text{カテゴリ } C \text{ における語彙数}} = \frac{\text{単語 } W_i\text{の数} + 1}{\text{カテゴリ } C \text{ における語彙数} + 1 \times \text{全語彙数}} \tag{5}$$

ツイート本文から単語を抽出し、上記した数式を利用すると本文がスパムか非スパムかに分類される。

4.6 形態素解析器

ツイート本文からスパムとなり得る要素を取り出す際には、文章構造を解析するための形態素解析器が必要となる。本研究では、形態素解析器に MeCab [17] を利用している。MeCab によってツイート本文を単語毎に分解し、その品詞を読み取る。MeCab の動作例を図 7 へ示す。

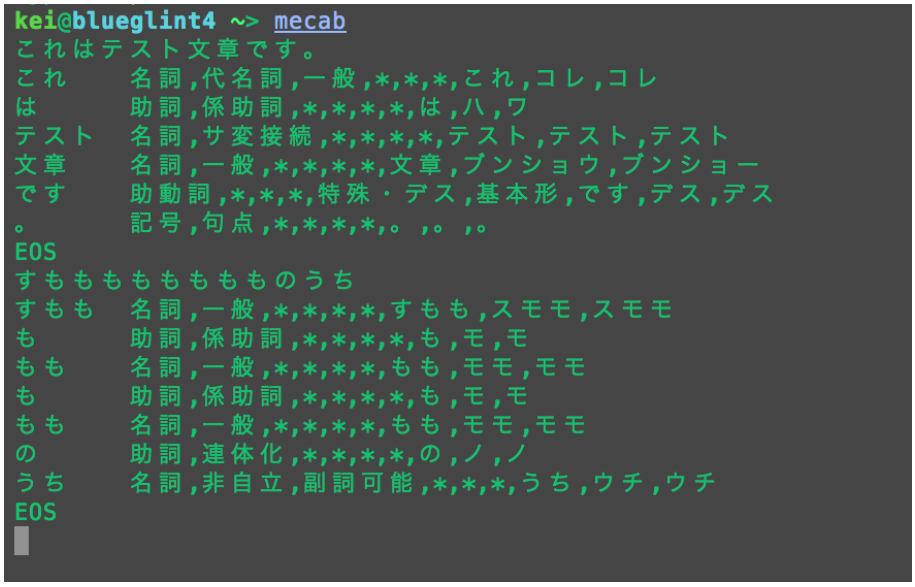

図 7　MeCab の動作画面。この図では「これはテスト文章です」「すももももももものうち」という文章を入力し、形態素解析を行った結果が出力されている。オプションを指定することで出力される情報を変更できる。

ソーシャルメディアのリアルタイム性を考えた場合、形態素解析を行う際の動作時間が長く掛かってしまうのは避けるべき点である。また、他の解析器には ChaSen [18] や JUMAN [19] などが存在する。通例的に MeCab は、他の解析器と比較して動作が高速に行われるとされており、本システムで採用した。更に本研究では標準の辞書に加え、流行語やネット言葉などに対応できる辞書 (mecab-ipadic-neologd) [20] を追加し、不快な要素を特定しやすくした。図 8, 図 9 に標準の MeCab と mecab-ipadic-neologd の動作比較例を示す。

また、スパム判定に不要な要素が本文中に入っている場合、フィルタへ要素追加する際に入ってしまう可能性がある。本研究では「@ユーザ名」、「RT」、「URL」が該当する。これらはツイートを取得する際に付属するデータである。そこで本実装では、テーブルへ単語を入力する際に上記の要素を含まないようにした。これにより、ツイート本文内の不快要素を効率良く取得できるものとした。

```
kei@blueglint4 ~> mecab
弊社ではC#やVisual Basicを活用し、ソフトウェアを開発しています。
弊社	名詞,一般,*,*,*,*,弊社,ヘイシャ,ヘイシャ
で	助詞,格助詞,一般,*,*,*,で,デ,デ
は	助詞,係助詞,*,*,*,*,は,ハ,ワ
C	名詞,固有名詞,組織,*,*,*,*
#	名詞,サ変接続,*,*,*,*,*
や	助詞,並立助詞,*,*,*,*,や,ヤ,ヤ
Visual	名詞,一般,*,*,*,*,*
Basic	名詞,一般,*,*,*,*,*
を	助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
活用	名詞,サ変接続,*,*,*,*,活用,カツヨウ,カツヨー
し	動詞,自立,*,*,サ変・スル,連用形,する,シ,シ
、	記号,読点,*,*,*,*,、,、,、
ソフトウェア	名詞,一般,*,*,*,*,ソフトウェア,ソフトウェア,ソフトウェア
を	助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
開発	名詞,サ変接続,*,*,*,*,開発,カイハツ,カイハツ
し	動詞,自立,*,*,サ変・スル,連用形,する,シ,シ
て	助詞,接続助詞,*,*,*,*,て,テ,テ
い	動詞,非自立,*,*,一段,連用形,いる,イ,イ
ます	助動詞,*,*,*,特殊・マス,基本形,ます,マス,マス
。	記号,句点,*,*,*,*,。,。,。
EOS
```

図 8　標準の MeCab で「弊社では C#や Visual Basic を活用し、ソフトウェアを開発しています。」を形態素解析した場合の図。C#, Visual Basic という専門用語を単語として読み取れていない。

```
kei@blueglint4 ~>
mecab -d /usr/local/Cellar/mecab/0.996/lib/mecab/dic/mecab-ipadic-neologd/
弊社ではC#やVisual Basicを活用し、ソフトウェアを開発しています。
弊社	名詞,一般,*,*,*,*,弊社,ヘイシャ,ヘイシャ
で	助詞,格助詞,一般,*,*,*,で,デ,デ
は	助詞,係助詞,*,*,*,*,は,ハ,ワ
C#	名詞,固有名詞,一般,*,*,*,C#,シーシャープ,シーシャープ
や	助詞,並立助詞,*,*,*,*,や,ヤ,ヤ
Visual Basic	名詞,固有名詞,一般,*,*,*,Visual Basic,ビジュアルベーシック,ビジュアルベーシック
を	助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
活用	名詞,サ変接続,*,*,*,*,活用,カツヨウ,カツヨー
し	動詞,自立,*,*,サ変・スル,連用形,する,シ,シ
、	記号,読点,*,*,*,*,、,、,、
ソフトウェア	名詞,一般,*,*,*,*,ソフトウェア,ソフトウェア,ソフトウェア
を	助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
開発	名詞,サ変接続,*,*,*,*,開発,カイハツ,カイハツ
し	動詞,自立,*,*,サ変・スル,連用形,する,シ,シ
て	助詞,接続助詞,*,*,*,*,て,テ,テ
い	動詞,非自立,*,*,一段,連用形,いる,イ,イ
ます	助動詞,*,*,*,特殊・マス,基本形,ます,マス,マス
。	記号,句点,*,*,*,*,。,。,。
EOS
```

図 9　mecab-ipadic-neologd を用いて「弊社では C#や Visual Basic を活用し、ソフトウェアを開発しています。」を形態素解析した場合の図。標準の辞書とは異なり、C#, Visual Basic という単語を認識している。

4.7 スパムフィルタの検証

テストコードを用いて、実装したスパムフィルタが想定通り文書を判定するか信頼度を検証した。検証内容は以下のようにした。

- 検証法はホールドアウト法を用いる
- 判定用の文書は Twitter 上のツイートから引用
- 判定させる文書はスパム文書 5 文、非スパム文書 5 文の合計 10 文
- 判定基準の文書はスパム文書 25 文、非スパム文書 25 文の合計 50 文
- スパムをきちんとスパムと判定できている確率を確認する (正答率)
- 非スパムをスパムと判定してしまう確率を確認する (誤判定率)

検証結果は以下の通りである。文書は、数値が 0 に近い項目 (Spam, Ham) にカテゴライズされる。

4.7.1 ホールドアウト法

ホールドアウト法にて検証を行った結果を以下に示す。

スパム正答率

1. {"Spam"=>-101.18310267362918, "Ham"=>-103.52125417887535}
2. {"Spam"=>-94.74505457672164, "Ham"=>-94.84511949353194}
3. {"Spam"=>-105.47513473823419, "Ham"=>-105.67203935093477}
4. {"Spam"=>-118.36803805036566, "Ham"=>-118.67894197923931}
5. {"Spam"=>-169.87242282562562, "Ham"=>-170.6481572947729}

正答率 100%

非スパム誤判定率

1. {"Spam"=>-94.7786688133544, "Ham"=>-92.69433432147252}
2. {"Spam"=>-99.07070087795942, "Ham"=>-97.02510226443366}
3. {"Spam"=>-193.4954062992695, "Ham"=>-192.30199700957854}
4. {"Spam"=>-129.11492533019455, "Ham"=>-127.34047786516157}
5. {"Spam"=>-96.92468484565691, "Ham"=>-94.85971829295309}

誤判定率 0%

スパム正答率、非スパム誤判定率共に良好な数値を算出していることが確認できた。

4.7.2 機械学習手法について

本研究ではナイーブベイズ分類器を用いているが、機械学習の分野において自然言語処理を行う手法が多数存在している。中でもサポートベクターマシン (SVM) は、2012 年の研究 [21] において日本語コーパス[*6]を用いた Spam/Ham 判定に適しているという結果が出ている。ただし SVM の弱点として、サンプル数が多い場合に最適化計算が膨大になること、学習速度が非常に遅くなるというものが考えられる。本研究のように、ソーシャルメディアという情報量=サンプル数の多いプラットフォームの場合は、今後どういった結果が見られるのか確認すべきと考える。そのため、続けての検証が必要であると考える。

[*6] 日本語や英語のような特定の言語、または複数の言語からなるテキストデータの集まりのこと。

5　結果と考察

本研究を行った上で、筆者がシステムを利用し、確認した動作について記述する。また利点と問題点について自説を述べる。

5.1　システムの動作

既存の Twitter クライアントでは、あるユーザそのものやキーワードマッチングによってツイートの非表示を行っていた。本システムでは、ユーザがシステムへ「不快である」= スパムとするツイート内容を学習させることで、システムが自動的にツイートを判定し、マスクを掛けるようにしている。著者が実際にシステムを利用してみたところ、問題ないと判断されたツイートが表示され、スパムと思い学習させた内容に沿ってマスクを掛けていることが確認できた。

5.2　利点

5.2.1　マスク

実際にタイムラインへ流れてきたツイートに対し、スパム/ニュートラルの入力を行った結果、新規ツイート読み込み時に不快な内容を含んでいると判断したツイートをマスクした。

5.2.2　ユーザの志向性把握

本研究によって、ユーザが好まないとする要素を汲む土台が出来上がった。

5.3　問題点

5.3.1　API

本研究では、Twitter REST API を利用している。しかし、公式の意向は REST API から Streaming API の利用を推奨している。

5.3.2　スムージング

本実装ではラプラススムージングを用いて判定用スコアの調整を行った。他のスムージング手法を用いてフィルタリングを行うことで、より精度が向上する可能性がある。

5.3.3　崩れた表現

本実装では MeCab に辞書を追加することで、デフォルトの MeCab では認識できない単語を読み取れるようにした。しかし、ソーシャルメディアでは日本語の文法通りでない口語体の文章や言葉本来の表記にしないような「崩れた表現」が多用されている。それにより、投稿内容からの情報抽出が難しくなっている。

6　今後の改善点

今後の改善点として考えられるのは以下の事項である。

6.1　判定用データのテンプレート製作

利用初期段階では判定用のデータが不足しており、期待される動作にならない可能性がある。それに対し、テンプレートとなる DB を用意しておくことで問題が緩和されると考えた。あらかじめ数種類のテンプレートを用意しておき、ユーザに選択させて利用を開始してもらう方が、よりスムーズにシステムの評価が可能である。

6.2　スムージング手法切り替えの実装

ラプラススムージングとは別のスムージング手法を検証する。それによりソーシャルメディアに沿った形のスムージング手法が求められる。

6.3　他フィルタとの組み合わせ

4.4 で述べたようなフィルタと、本研究で用いているベイジアンフィルタを併用することで、フィルタの有効性が向上すると考える。例としては、ユーザ毎に信頼度を設定するなどが挙げられる。

6.4　形態素解析器のソーシャルメディア向け改良

ソーシャルメディアでは「崩れた表現」が多く、自然言語処理を用いたデータ解析が難しい。そうしたソーシャルメディア特有の表現方法には、形態素解析器をソーシャルメディア向けに改良することで解決できると考える。先行研究として、日本語形態素解析器「MoCA」[22] が存在する。

6.5　スマートフォン用アプリとして移植

本実装では Web ブラウザ用の Twitter クライアントを製作したが、ターゲットとする若年層が主に利用するデバイスはスマートフォンやタブレット端末である [23]。そこを考慮すると、スマートフォン向けのネイティブアプリとして製作し、日常的に扱えるものにすべきと考える。

7　まとめ

本研究では、ユーザが好まないであろうツイートへ自動的にマスクを掛け、対人間のコミュニケーションギャップを防ぐ支援を行った。利用初期段階では、スパム/ニュートラル判定に用いるデータが少なく、効果的なフィルタリングが確認できていない。しかしながら、用いるデータが増加していくことでフィルタの精度が向上することを考慮すると、長期的に見て効果が期待できる。このシステムを多くのユーザが日常的に利用することで、不要な争いを減らすことができ、コミュニケーションが円滑に行える。結果的に炎上の防止に繋がることが期待できる。

謝辞

本研究を実施するにあたり、大垣斉准教授には大学生活の様々な面において御指導御鞭撻を戴きました。また、当研究室の OB である水野貴弘氏、湯川正洋氏には、研究を進める上での考察や製作手法について多くの御指導を戴きました。そして、team.andrew に所属している卒研生、ゼミ生、居候の方々には研究内容のみならず生活面につきましても御助言を戴きました。ここに感謝の意を示し、謝辞と致します。

参考文献

[1] 総務省. 平成 26 年度 情報通信白書. 総務省, 2014.

[2] 総務省. 平成 27 年度 情報通信白書. 総務省, 2015.

[3] 小林直樹. ソーシャルメディア炎上事件簿. 日経 BP 社, 2011.

[4] 坂下哲浩. Sns における発言のしやすさと態度形成 ソーシャルメディアにおける炎上から . Master's thesis, 慶應義塾大学大学院, 2012.

[5] 岩崎昭彦. Cgm における炎上の分析とその応用. 人工知能学会論文誌 30 巻 1 号, 2015.

[6] 中川淳一郎. ウェブを炎上させるイタい人たち——面妖なネット原理主義者の「いなし方」. 宝島社, 2010.

[7] Blake Mizerany. Sinatra. http://www.sinatrarb.com/.

[8] Norman Clarke. Haml. http://haml.info/.

[9] Twitter Inc. Rest apis — twitter developers. https://dev.twitter.com/rest/public.

[10] Incept Inc. ベイジアンフィルタとは | bayesian filter - 意味/解説/説明/定義 ：it 用語辞典. http://e-words.jp/w/%E3%83%99%E3%82%A4%E3%82%B8%E3%82%A2%E3%83%B3%E3%83%95%E3%82%A3%E3%83%AB%E3%82%BF.html.

[11] 小島國照. 第 4 回：押さえておきたいスパムフィルタの技術. https://thinkit.co.jp/free/article/0611/8/4/.

[12] Twitter Inc. ミュート機能を追加、twitter の使い方が広がります — twitter blogs. https://blog.twitter.com/ja/2014/0513mute, 2014.

[13] Twitter Inc. Twitter ユーザーのブロック — twitter ヘルプセンター. https://support.twitter.com/articles/238371, 2015.

[14] Twitter Inc. スパム報告ツールを追加 — twitter blogs. https://blog.twitter.com/ja/2009-18, 2009.

[15] Paul Graham. A plan for spam. http://www.paulgraham.com/spam.html, 2002.

[16] 山口和紀. computing machinery. https://lecture.ecc.u-tokyo.ac.jp/~yamaguch/keisan-kiko-ron-1/2008/bayes.html.

[17] 工藤拓. Mecab: Yet another part-of-speech and morphological analyzer. http://taku910.github.io/mecab/.

[18] 奈良先端科学技術大学院大学松本研究室. chasen legacy – an old morphological analyzer. http://chasen-legacy.osdn.jp/, 2007.

[19] 京都大学黒橋・河原研究室. Juman - kurohashi-kawahara lab. http://nlp.ist.i.kyoto-u.ac.jp/index.php?cmd=read&page=JUMAN&alias%5B%5D=%E6%97%A5%E6%9C%AC%E8%AA%9E%E5%BD%A2%E6%85%8B%E7%B4%A0%E8%A7%A3%E6%9E%90%E3%82%B7%E3%82%B9%E3%83%86%E3%83%A0JUMANhttp://nlp.ist.i.kyoto-u.ac.jp/index.php?JUMAN, 2015.

[20] Sato Toshinori. Neologism dictionary based on the language resources on the web for mecab, 2015.

[21] 藤森夏輝. Spam メールの判別に適した機械学習. 平成 24 年度学士学位論文, 2012.

[22] 利根川翔. 日本語形態素解析における崩れた表記への対応手法の提案と評価. Master's thesis, 早稲田大学大学院 基幹理工学研究科 情報理工学専攻, 2012.

[23] モバイルマーケティング研究所. 10 代は twitter が人気、その使い方とは？スマホ利用実態調査 — モバイルマーケティング研究所 —moduleapps. https://moduleapps.com/mobile-marketing/teen-research/, 2015.

付録 A　ソースコード

A.1　main.rb(基礎部分)

```
# coding: utf-8

### require gems
require 'rubygems'
require 'sinatra'
require 'sinatra/reloader'
require 'twitter'
require 'haml'
require 'sqlite3'
require 'active_record'
require 'sinatra/activerecord'
require 'json'
require 'omniauth'
require 'omniauth-twitter'
require 'dotenv'
require 'addressable/uri'

require_relative 'bayesian_filter'

### SQLite3 との接続準備
ActiveRecord::Base.establish_connection(
  'adapter' => 'sqlite3',
  'database' => './db/bayes_tweets.db'
)

### スパムテーブルと接続
class Spam_tweet < ActiveRecord::Base
  self.table_name = 'spam_tweets'
end

### ニュートラルテーブルと接続
class Ham_tweet < ActiveRecord::Base
  self.table_name = 'ham_tweets'
end

### .env に入れているアプリ用キーを読みこむ
Dotenv.load
```

```
### sinatra のエラーログ出力
set :environment, :development     #=> localhost 以外から参照可能にする
set :dump_errors, true       #=> STDERR に log 出力
set :show_exceptions, true  #=> development モードでデバッグ表示

### sinatra その他設定
set :server, :thin #=> thin を Web サーバとして運用
set :sessions_secret, DateTime.now.to_s #> トークン取得用のセッションを張る

### sinatra の本設定
configure do
  enable :sessions

  use OmniAuth::Builder do
    provider :twitter, ENV['CONSUMER_KEY'], ENV['CONSUMER_SECRET']
  end
end

### DB からレコードを引っ張りスパムフィルタ生成
# DB のレコードを変数へ入れる
pull_spam    = Spam_tweet.all
pull_ham     = Ham_tweet.all

# スパム部分の学習を行う
pull_spam.each do |record|
  record.cnt.times do |t|
    Bayesian_filter.train('Spam', record.word)
  end
end
# ニュートラル部分の学習を行う
pull_ham.each do |record|
  record.cnt.times do |t|
    Bayesian_filter.train('Ham', record.word)
  end
end

helpers do
  def logged_in?
    session[:twitter_oauth]
  end

  ### 上記のキーを元に REST API を叩いて config に落とし込む
  ### クライアントの認証用に client を使う
```

```
  def client
    Twitter::REST::Client.new do |config|
      config.consumer_key           = ENV['CONSUMER_KEY']
      config.consumer_secret        = ENV['CONSUMER_SECRET']
      config.access_token           = session[:twitter_oauth][:token]
      config.access_token_secret    = session[:twitter_oauth][:secret]
    end
  end
end

### 認証ができていない場合に認証ページへ飛ばす
before do
  pass if request.path_info =~ /^\/auth\//
  redirect to('/auth/twitter') unless logged_in?
end

### ツイートを扱うために整えておく
class Tweet
  ### アクセサを作っておく
  attr_accessor :name, :screen_name, :full_text, :is_mask, :profile_img
  ### 初期化
  def initialize(name, screen_name, full_text, is_mask, profile_img)
    @name        = name #=>ここにユーザ名
    @screen_name = screen_name #=>ここにスクリーン名
    @full_text   = full_text #=>ここにツイート本文
    @is_mask     = is_mask   #=>これを元に表示・NG かどうか判別
    @profile_img = profile_img #=>アイコンの URL 保持
  end
end

### 配列に SN と本文を入れ込む
def get_tweets()
  tweets = Array.new() #=>ツイートを入れ込む配列変数
  client.home_timeline.map do |tweet|
    is_mask = Bayesian_filter.judge(tweet.full_text)
    tweets.push(Tweet.new(tweet.user.name, tweet.user.screen_name, tweet.full_text, is_mask, tweet.user
  end
  return tweets
end

### OAuth 認証からのコールバック
get '/auth/twitter/callback' do
  session[:twitter_oauth] = env['omniauth.auth'][:credentials]
```

```
  redirect to ('/')
end

### OAuth 認証失敗
get '/auth/failure' do
  puts 'authentication failed'
end

### sinatra エラー処理
error do |e|
  status 500
  body e.message
end

### トップページ
get '/' do
  @oauth = session[:twitter_oauth]
  @tweets = get_tweets()
  @tweets.each do |twe|
    if !twe.full_text.nil?
      Bayesian_filter.register_tweet(Ham_tweet, twe.full_text.gsub(/https?:\/\/.*/,'').gsub(/@.*:/,'').
      Bayesian_filter.add_ham_train(twe.full_text.gsub(/https?:\/\/.*/,'').gsub(/@.*:/,'').gsub(/RT\s/,
    end
  end
  haml :index
end

### スパムの追加はこちら
post '/add/spam' do
  spam_params = @params[:full_text]
  if !spam_params.nil?
    Bayesian_filter.register_tweet(Spam_tweet, spam_params.gsub(/https?:\/\/.*/,'').gsub(/@.*:/,'').gsu
    Bayesian_filter.add_spam_train(spam_params.gsub(/https?:\/\/.*/,'').gsub(/@.*:/,'').gsub(/RT\s/,'')
  end
end

### ニュートラルの追加はこちら
post '/add/ham' do
  ham_params = @params[:full_text]
  if !ham_params.nil?
    Bayesian_filter.register_tweet(Ham_tweet, ham_params.gsub(/https?:\/\/.*/,'').gsub(/@.*:/,'').gsub(
    Bayesian_filter.add_ham_train(ham_params.gsub(/https?:\/\/.*/,'').gsub(/@.*:/,'').gsub(/RT\s/,''))
  end
```

```
end

get '/logout' do
  session.clear
  redirect to '/'
end

### 処理後に DB 接続を解除
after do
  ActiveRecord::Base.connection.close
end
```

A.2 bayesian_filter.rb(ベイジアンフィルタ処理部分)

```
# coding: utf-8

require 'rubygems'
require 'mecab'
require 'classifier'

### classifier gem の一部を書き換え
class Classifier::Bayes
  def classifications(text)
    score = Hash.new
    training_count = @category_counts.values.inject { |x,y| x+y }.to_f
@categories.each do |category, category_words|
score[category.to_s] = 0
total = category_words.values.inject(0) {|sum, element| sum+element}
text.word_hash.each do |word, count|
s = category_words.has_key?(word) ? category_words[word] : 0.1
score[category.to_s] += Math.log(s+1 / total.to_f)
end
      s = @category_counts.has_key?(category) ? @category_counts[category] : 0.1
      score[category.to_s] += Math.log(s+1 / training_count)
end
return score
  end
end

module Bayesian_filter
  @@bayes  = Classifier::Bayes.new('Spam', 'Ham')
  @@wakati = MeCab::Tagger.new('-d /usr/local/Cellar/mecab/0.996/lib/mecab/dic/mecab-ipadic-neologd')
```

```
### 判定に用いる品詞を抽出
def self.addable_node(text)
  /^形容詞|^形容動詞|^感動詞|^副詞|^連体詞|^名詞|^動詞/ =~ text
end

### 単語の配列を生成
def self.create_word_array(node)
  word_array = Array.new()
  while node
    feature = node.feature.force_encoding('UTF-8')
    word_array << node.surface.force_encoding('UTF-8') if addable_node(feature)
    node = node.next
  end
  return word_array
end

### テーブルから単語を検索する
def self.find_by_word(table, word)
  table.where(word: word)
end

### DB の単語登録回数を加算する
def self.update_count(record)
  record.cnt += 1
  record.save
end

### DB に存在しない単語を登録
def self.insert_word(table, word)
  table.create(word: word, cnt: 1)
end

### DB にスパムツイートの単語と出現回数を入れ込む
def register_tweet(table, input_tweet)
  word_array = create_word_array(@@wakati.parseToNode(input_tweet))
  word_array.each do |word|
    record = find_by_word(table, word)
    if record.present?
      update_count(record.first)
    else
      insert_word(table, word)
    end
  end
```

```
  end
  module_function :register_tweet

  ### スパムをフィルタに学習させる
  def add_spam_train(input_tweet)
    word_array = create_word_array(@@wakati.parseToNode(input_tweet))
    word_array.each do |word|
      train('Spam', word)
    end
  end
  module_function :add_spam_train

  ### 非スパムをフィルタに学習させる
  def add_ham_train(input_tweet)
    word_array = create_word_array(@@wakati.parseToNode(input_tweet))
    word_array.each do |word|
      train('Ham', word)
    end
  end
  module_function :add_ham_train

  ### フィルタ形成用スパム単語入れ
  def train(category, spam_word)
    @@bayes.train(category, spam_word)
  end
  module_function :train

  ### 受け取った tweet がスパムか否かを判断させる部分
  ### 元のファイルだと今までの学習内容（ファイル中に置いたものだが）から判断させている
  def judge(current_tweet)
    compare = @@bayes.classifications(@@wakati.parse(current_tweet))
    result  = @@bayes.classify(@@wakati.parse(current_tweet))

    if compare['Spam'].infinite? || compare['Ham'].infinite?
      return false
    end

    puts compare
    puts result
    return result == 'Spam'
  end
  module_function :judge
end
```

A.3 index.haml(**クライアント画面部分**)

```
!!! 5
%html{lang: "ja"}
%head
  %meta{charset: "utf-8"}
  %title KLIENT
  %link{rel: "stylesheet", href: "/css/bootstrap.min.css"}
  %script{src: "/js/jquery-2.1.4.min.js"}
  %script{src: "/js/ajax_io.js"}
%body
  %div{class: "container"}
    %div{class: "row"}
      %table{class: "table table-striped table-bordered table-hover", style: "width: 100%;"}
        %tr
          %th{class: "col-sm-2"} ユーザ
          %th{class: "col-sm-7"} ツイート
          %th{class: "col-sm-3"} スパム報告
        - @tweets.each_with_index do |tw, index|
          %tr
            %td{class: "col-sm-2"}
              %img{src: "#{tw.profile_img}"}
              %div{style: "font-size:11px;"} #{tw.name}
              %div{style: "font-size:11px;"} @#{tw.screen_name}
            - if (!tw.is_mask)
              %td{class: "col-sm-7"}
                %div #{tw.full_text}
            - else
              %td{class: "col-sm-7 warning", onclick: "unmask(#{index})"}
                %div{class: "masked#{index}", style: "color:red;"} <マスクされています>
            %td{class: "col-sm-3"}
              %input{type: "hidden", id: "full_text#{index}", name: "full_text", value: "#{tw.full_text
              - if (!tw.is_mask)
                %button{class: "btn btn-danger", type: "button", onclick: "spam_filter(#{index})"} ス
パムです
              - else
                %button{class: "btn btn-success", type: "button", onclick: "neutral_filter(#{index})"}
パムじゃない
                %button{class: "btn btn-danger", type: "button", onclick: "spam_filter(#{index})"} ス
パムで合ってます
```