効率的なテキストストリーム処理のための自己適応的分類器

A Self-adaptive Classifier for Efficient Text-Stream Processing

吉永 直樹 [1*]　喜連川 優 [2,1]
Naoki Yoshinaga[1]　Masaru Kitsuregawa[2,1]

[1] 東京大学生産技術研究所
[1] Institute of Industrial Science, the University of Tokyo
[2] 国立情報学研究所
[2] National Institute of Informatics

Abstract: This paper presents a self-adaptive classifier for efficient text stream processing. Our method speeds up a classifier trained with many conjunctive features while the classifier solves the classification problems in processing a given text stream. The key idea is to keep and reuse classification results for fundamental classification problems to solve the forthcoming classification problems. We explore two enumeration schemes, the least frequently used (LFU) and the least recently used (LRU), to select fundamental classification problems while processing the text stream. Experimental results with a Twitter stream on the day of the 2011 Great East Japan Earthquake confirmed that the proposed method accelerated a classifier for a state-of-the-art deterministic dependency parser by a factor of up to 5.

1　はじめに

Twitter に代表されるマイクロブログの普及により，世間一般の人々が自身の体験や考えを気軽に発信する時代となっている．人々がマイクロブログを通じて発信する情報は，いまや実世界のあらゆる時間・空間を網羅するようになりつつあり，これをリアルタイムで解析することできれば，自然災害の状況把握に基づく減災，また企業や自治体，公的機関が提供する商品やサービス，施策の問題への迅速な対応など，より良い社会の実現に繋がることが期待できる．

しかしながら，現在の自然言語処理技術でマイクロブログのテキストストリームを解析しようとする場合，テキストストリームの質（内容）・量（単位時間あたりの流量）が時間的に変化するという性質から，解析の頑健性とリアルタイム性を両立させることは困難となっている [1, 2]．現在のところ，既存研究では，表記ゆれの正規化 [3] や未知語処理 [4] など，主に質の多様性に焦点を当てて研究が行われている．

本稿では，テキストストリームを言語解析する上で問題となる上記の問題のうち特に「量」の変化に焦点を当て，自然言語処理で広く用いられる組み合わせ素性に基づく線形分類器を，解析対象の分類問題に対して適応的に高速化させる手法を提案する．提案手法ではテキストストリームの流量が増えるときに，地震，大雪など自然災害の発生 [5] やイベントの実況など，特定の話題に関する投稿が増えるという観察[1]に基づき，類似する内容の文が多数発信されるという仮定のもと各分類問題に共通する基本分類問題を動的に列挙し，その分類結果を再利用することで分類器を適応的に高速化する．

実験では，東日本大震災発生当日のテキストストリームを係り受け解析し，その際に生成される分類問題（のストリーム）に対して提案手法を適用することで，提案手法の有効性を評価する．

2　関連研究

単語分割や品詞解析，基本句同定や係り受け解析，意味役割付与など多くの基礎言語解析において，その基本となる解析単位は文であり，効率的な解析アルゴリズムの多くは文単位での解析を高速化することを目的としている [6, 7, 8]．以下では，与えられたテキスト（文の集合）に対して適応的に解析を高速化する手法を紹介する．

Yoshinaga と Kitsuregawa [9, 10] は，自然言語処理で多用される組み合わせ素性に基づく分類器による分類を，事前に列挙した基本分類問題の分類結果を利用

*連絡先：東京大学生産技術研究所
〒 153-8505 東京都目黒区駒場 4-6-1
E-mail: ynaga@tkl.iis.u-tokyo.ac.jp

[1] https://2011.twitter.com/ja/tps.html

することで高速化する手法を提案している．この手法では，基本句同定タスクと係り受け解析タスクにおいて最大 3〜10 倍程度の高速化が可能であったと報告している．一方で，Srikumar, Kundu, Roth ら [11, 12] は，構造分類タスクを整数計画問題として解く際に，事前に解いた線形計画問題をデータベースに蓄えておき，これを結果として再利用することで解析を高速化する手法を提案している．彼らは，意味役割付与タスクと関係抽出タスクにおいて，最大 3 倍程度の高速化を達成したと報告している．これらの手法は言語処理タスクの入力の冗長性を利用した手法であり，それぞれ線形分類器，あるいは整数計画ソルバーが適用可能なタスク一般に利用できる手法であるが，動的に内容が変化するテキストストリームを解析対象とした場合の有効性は明らかでない．

van Noord は構文解析タスクにおいて，事前に大量のテキストを解析することで最終的な構文構造に貢献しにくい導出過程を収集し，そのような導出過程を解析時に読み飛ばすことで近似的に解析速度を向上させる手法を提案している．実験結果では，解析精度を保ったまま最大 4 倍程度の高速化を達成したと報告しているが，解析対象のドメインによって効果に違いがあることも合わせて報告されており，事前に解析対象のテキストが分からない状況下では適切な運用は難しいと考えられる．

Wachsmuth らは，複数の言語解析をパイプラインで組み合わせた情報抽出システムにおいて，システムを構成する各解析のスケジュールを調整することで全体の処理を効率化できることを示した [13]．彼らはさらに，自己学習法に基づく分類器で解析時間を予測することで与えられたテキストに対してパイプラインのスケジュールを動的に切り替える高速化手法を提案している [14]．彼らの手法とパイプラインを構成する各解析を適応的に効率化する我々の手法は相補的な関係にあり，組み合わせることで相乗的な効果が期待できる．

本稿では，Yoshinaga と Kitsuregawa が提案した組み合わせ素性に基づく分類器のための分類手法 [9] を応用し，テキストストリームから動的に基本分類問題を列挙することで適応的に分類器を高速化する手法を提案する．次節で手法 [9] を簡単に説明した後，4 節で提案手法について述べる．

3 背景知識

本節では，提案手法の土台となる線形分類器の高速化手法 [9] を説明する．この手法では，言語処理で多用される組み合わせ素性に基づく線形分類器（または多項式カーネルに基づく非線形分類器 [15]）を高速化の対象としている．

パーセプトロンや SVM などマージンに基づく二値分類器では与えられた分類問題 $\boldsymbol{x}$ のラベル $\boldsymbol{y} \in \{+1, -1\}$ を以下の式を用いて決定する（簡単のため，バイアス項は省略して議論を進める）．

$$m(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}) (= \sum_i w_i \phi_i(\boldsymbol{x})) \tag{1}$$

$$y = \begin{cases} +1 & (m(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{w}) \geq 0) \\ -1 & (m(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{w}) < 0). \end{cases} \tag{2}$$

ここで $m(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{w})$ は分離平面からのマージン，ϕ_i は素性関数であり，w_i は学習により得られた ϕ_i の重みである．

自然言語処理では文書分類など一部のタスクを除き，素性関数として言語的制約などを表現する指示関数（二値関数）を用いることが多い．本稿では {0,1} を返す二値関数のみを素性関数として想定し，以後マージン $m(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{w})$ を以下の式

$$m(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{w}) = \sum_{i \in \{j | \phi_j(\boldsymbol{x}) = 1\}} w_i \tag{3}$$

で表現する．また，素性関数 ϕ_i を単に素性と呼び，$\boldsymbol{x}$ について $\phi_i(\boldsymbol{x}) = 1$ となるとき，$\boldsymbol{x}$ が素性 ϕ_i を含む，あるいは $\boldsymbol{x}$ で素性 ϕ_i が発火する，と表現する．式 3 は $|\boldsymbol{\phi}(\boldsymbol{x})|$ を発火素性数として $\mathcal{O}(|\boldsymbol{\phi}(\boldsymbol{x})|)$ で計算することができる．

既存手法 [9] では，分類タスクに共通して出現する基本分類問題 $\boldsymbol{x}_c$ に対して分類結果 $M_{\boldsymbol{x}_c} \equiv m(\boldsymbol{x}_c; \boldsymbol{\phi}, \boldsymbol{w})$ を事前に計算しておき，これを入力の分類問題の部分結果として用いることで分類を高速化する．

$$m(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{w}) = M_{\boldsymbol{x}_c} + \sum_{i \in \{j | \phi_j(\boldsymbol{x}) = 1, \phi_j(\boldsymbol{x}_c) = 0\}} w_i \tag{4}$$
$$where \quad \forall i \in \{j | \phi_j(\boldsymbol{x}_c) = 1\}, \phi_i(\boldsymbol{x}) = 1.$$

式 4 が式 3 より高速に計算できるためには，$M_{\boldsymbol{x}_c}$ が $\mathcal{O}(|\boldsymbol{\phi}(\boldsymbol{x}_c)|)$ より高速に取得できる必要があるが，$\boldsymbol{x}_c$ に組み合わせ素性 $\phi_{i,j}(\boldsymbol{x}) = \phi_i(\boldsymbol{x})\phi_j(\boldsymbol{x})$ が含まれる状況下では，その組み合わせ素性 $(\phi_{i,j})$ を構成する基本素性 (ϕ_i, ϕ_j) のみ確認すれば $M_{\boldsymbol{x}_c}$ を取得可能であるため，高速化できる．例えば，$\boldsymbol{\phi} = \{\phi_i, \phi_j, \phi_{i,j}\}$ とするとき，$m(\boldsymbol{x}_c; \boldsymbol{\phi}, \boldsymbol{w})$ の計算には組み合わせ素性を含む 3 つの素性の重みを加算する必要があるが，$M_{\boldsymbol{x}_c}$ を取得するには 2 つの基本素性 ϕ_i, ϕ_j のみを確認すればよい．一般的な議論をすると，元の組み合わせ素性を含む素性空間 $\boldsymbol{\phi}$ に対して，組み合わせ素性を除いて基本素性のみに縮退した素性空間 $\boldsymbol{\phi}'$ を考えれば，マージンを得るために確認する素性の数を多項式オーダから線形オーダに落とすことができる．

なお，式 4 による高速化の効果を最大とするためには $\boldsymbol{x}$ でのみ発火する素性 $\{\phi_i | \phi_i(\boldsymbol{x}) = 1, \phi_i(\boldsymbol{x}_c) = 0\}$

の数がなるべく少なくなるような（直感的に言い換えるとなるべく $\boldsymbol{x}$ に近い）基本分類問題 $\boldsymbol{x}_c$ のマージン $M_{\boldsymbol{x}_c}$ を部分結果として用いることが望ましい．しかしながら，全てのありうる $\boldsymbol{x}$ について $M_{\boldsymbol{x}} \equiv m(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{w})$ を事前計算しようとすると $\mathcal{O}(2^{|\boldsymbol{\phi}|})$ の記憶領域が必要となり，現実的ではない．そこで既存手法 [9] では，分類器を用いて解く言語処理タスクの入力を大量に用意して実際に解く分類問題を列挙し，それらに共通する部分分類問題を基本分類問題として採用する手法を提案している．基本分類問題の選択にあたっては，その出現頻度と式 4 において削減される計算コストの大きさに基づき，より有用性の高い基本分類問題を選択する．

抽出した基本分類問題 $\boldsymbol{x}_c$ は発火する素性のインデクス列（以下，素性列）で表現し，計算した分類結果 $M(\boldsymbol{x}_c)$ と合わせてトライに格納する．入力の分類問題になるべく近い基本分類問題を式 4 で利用するため，各基本素性の頻度を計算し，その頻度順で発火する基本素性を並べてトライに保存・検索する素性列を構成する．これにより，最長接頭辞検索により，入力の分類問題に類似する基本分類問題を $\mathcal{O}(|\boldsymbol{\phi}'(\boldsymbol{x}_c)|)$ で取得することが可能となる．

4 提案手法

前節で述べた高速分類手法 [9] では，部分結果として利用する基本分類問題は事前に列挙することを想定していた．本研究では，基本分類問題を動的に列挙することで，テキストストリーム[2]に対して適応的に分類を高速化する手法を提案する．これを実現するにあたり解くべき課題は，1) マージンを保持する基本分類問題をどう選ぶか, また，2) 基本分類問題をどのようなデータ構造で管理にするか，という 2 点である．以下で，これらの課題に対する本研究での解を順に述べる．

4.1 基本分類問題の動的列挙

既存手法 [9] では分類問題の頻度と計算の削減コストを考慮して基本分類問題を選択していたが，テキストストリームから分類問題を列挙する場合には頻度が動的に変化するため，どの分類問題が（テキストストリーム全体を通して）有用な基本分類問題となっているか事前に知ることはできない．従って，テキストストリームの変化に合わせて必要な分類問題は追加し，不要なものは削除するなどして基本分類問題の集合を適応的に更新する必要がある．

Algorithm 1 に我々の提案する基本分類問題の動的列挙に基づく適応的な分類手法を示す．提案手法では

[2]正確には，テキストストリームを言語解析する際に生成される分類問題のストリーム．

Algorithm 1 基本分類問題の動的列挙に基づく分類

Input: $\boldsymbol{x}$, $\boldsymbol{\phi}$, $\boldsymbol{\phi}'$, $\boldsymbol{w}$, $\mathcal{X}_c$, k
Output: $m(\boldsymbol{x}) \in \mathbb{R}$, $\mathcal{X}_c$
1: initialize: $\boldsymbol{x}_c$ *s.t.* $\boldsymbol{\phi}'(\boldsymbol{x}_c) = \mathbf{0}$, $m(\boldsymbol{x}) \leftarrow 0$
2: **repeat**
3: $\quad i = \underset{i \in \{j \mid \phi'_j(\boldsymbol{x})=1, \phi'_j(\boldsymbol{x}_c)=0\}}{\operatorname{argmax}} \textsc{freq}(\phi'_i)$
4: $\quad \phi'_i(\boldsymbol{x}_c) \leftarrow 1$
5: $\quad$ **if** $\boldsymbol{x}_c \notin \mathcal{X}_c$ **then**
6: $\quad\quad M_{\boldsymbol{x}_c} \leftarrow m(\boldsymbol{x}) + \sum_{i \in \{j \mid \phi_j(\boldsymbol{x})=1, \phi_j(\boldsymbol{x}_c)=0\}} w_i$
7: $\quad\quad$ **if** $|\mathcal{X}_c| = k$ **then**
8: $\quad\quad\quad \mathcal{X}_c \leftarrow \mathcal{X}_c - \{\textsc{useless}(\mathcal{X}_c)\}$
9: $\quad\quad \mathcal{X}_c \leftarrow \mathcal{X}_c \cup \{\boldsymbol{x}_c\}$
10: $\quad m(\boldsymbol{x}) \leftarrow M_{\boldsymbol{x}_c}$
11: **until** $\boldsymbol{\phi}'(\boldsymbol{x}_c) \neq \boldsymbol{\phi}'(\boldsymbol{x})$
12: **return** $m(\boldsymbol{x})$, $\mathcal{X}_c$

観測された分類問題 $\boldsymbol{x}$ から分類器の学習時に高頻度で観測された基本素性 ϕ'_i を順に取り出し，基本分類問題 $\boldsymbol{x}_c$ を漸進的に構成する（3-4 行目）．このようにして構成された $\boldsymbol{x}_c$ がその時点で保持する基本分類問題集合 $\mathcal{X}_c$ に含まれていた場合には以前に計算したマージン $M_{\boldsymbol{x}_c}$ を返し，そうでない場合は式 4 を用いてマージン $M_{\boldsymbol{x}_c} \equiv m(\boldsymbol{x}_c; \boldsymbol{\phi}, \boldsymbol{w})$ を計算した後，$\boldsymbol{x}_c$ を基本分類問題として $\mathcal{X}_c$ に追加する（9 行目）．

ここで重要なのは，基本分類問題数が予め定めた上限数 k に達した場合に，$\mathcal{X}_c$ に保存された基本分類問題から最もその後の分類の高速化に貢献しない考えられる基本分類問題を取り除く関数 USELESS である．本稿では，一般的なキャッシュアルゴリズムにおけるキャッシュの管理ポリシーを参考に以下の 2 つの基準を提案・比較する．

Least Frequently Used (LFU) テキストストリーム中で観測された分類問題の頻度を計測し，高頻度で参照される基本分類問題のみを保持するように低頻度の基本分類問題を削除する．

$$\textsc{useless}_{\textsc{lfu}}(\mathcal{X}_c) = \underset{\boldsymbol{x}_c}{\operatorname{argmin}} \textsc{freq}(\boldsymbol{x}_c) \quad (5)$$

テキストストリーム中での基本分類問題の頻度は space saving アルゴリズム [16] を用いて近似的に計測し，頻度計測の対象から外れた基本分類問題を削除する．

Least Recently Used (LRU) テキストストリームの流量が増えるときには，特定の話題に偏って情報が発信されることが多い．この点を考慮すると，より最近アクセスされた分類問題を基本分類問題として保持することも有効と考えられる．

$$\textsc{useless}_{\textsc{lru}}(\mathcal{X}_c) = \underset{\boldsymbol{x}_c}{\operatorname{argmin}} \textsc{time}(\boldsymbol{x}_c) \quad (6)$$

そこで，基本分類問題のアクセス時間を保持し，新しい分類問題が観測された際には最も過去に参照された基本分類問題を削除する．

基本分類問題の上限数 k については，分類開始時に初期値として固定するものとし，実験でその値を変化させてその影響を確認する．

Algorithm 1 は入力の分類問題に含まれる全ての基本素性を用いて基本分類問題を構成するため，低頻度の基本素性を含む基本分類問題に関して，追加と削除が頻繁に繰り返される可能性がある．そこで我々は，$\boldsymbol{x}$ にのみ含まれる基本素性を元に，マージンの変化の上限・下限の見積もりを行い，マージンの符号（分類問題のラベル）が変わらないことが分かり次第，計算を打ち切ることを試みる．具体的には，各基本素性について，その素性を含む組み合わせ素性の重みの上限・下限を記録しておき，これを素性数と掛け合わせることで，マージンの変化の上限・下限を $\mathcal{O}(1)$ で計算する[3]．

4.2 ダブル配列を用いた基本分類問題の管理

基本分類問題を管理するデータ構造としてはダブル配列 [17] に基づくトライを利用する．トライは基本分類問題のキーに対してそのインデクス $i(1 \leq i \leq k)$ を値として保持し，これを前節で述べた基本分類問題の列挙時に利用する．基本分類問題をダブル配列に基づくトライで管理することで，Algorithm 1 における検索・削除操作（それぞれ，5, 8 行目）が $\mathcal{O}(|\boldsymbol{\phi}'(\boldsymbol{x}_c)|)$ ではなく，$\mathcal{O}(1)$ で行えること[4]に注意されたい．

従来，ダブル配列は検索は高速であるものの，動的な更新は低速なデータ構造であると認識されている．しかしながら近年の研究 [19] により，適切に実装しさえすれば，更新についてもハッシュや HAT トライ [20] などに匹敵する速度で行うことが可能となっている[5]．我々は [19] に改良を施した動的ダブル配列を用いて基本分類問題の管理を行う．

ダブル配列を用いて基本分類問題の管理を効率的に行うためには，検索・更新・削除の際にアクセスするノードの数をなるべく少なくすることが重要である．そこで，以下のアルゴリズム・データ構造的な工夫によりトライのノード数を可能な限り削減することを試みた．

可変長バイト符号化を用いた素性表現の圧縮　トライ中における各基本分類問題の表現（素性列）を圧縮するため，素性には学習データ中での頻度に従って降順にインデクスを与え，頻度順に並び替える．さらに，情報検索における転置インデクスの圧縮で使われる差分リスト[6]で素性列を表現し，各インデクスを可変長バイト符号化 [21] により圧縮して連結することで，トライ中の素性列の表現を得る．これにより，トライ中のノード数を大きく削減することができる．

値ノードの縮約による動的ダブル配列の圧縮　一般的なダブル配列の実装ではキーに対する値は終端文字'\0'（あるいは'#' などキーに含まれない文字）を辿った先のノード（以下，値ノード）に埋め込むことが多い [9, 22]．しかしながら，トライ中で他のキーの接頭辞とならないキーについては，キーの末尾の文字を辿った先のノードに直接値を埋め込むことが可能である．このとき，終端であることを判別するため，値は負値で保存する．なお，他のキーの接頭辞となるキーに対応する値については，従来の実装と同様に，終端文字'\0' を辿った先のノードに格納する．

5 評価実験

本節では，係り受け解析の分類器に提案手法を適用し，Twitter のテキストストリームを解析対象として提案手法の有効性を検証する．以後の実験には Intel® Core™ i7-3720QM 2.6GHz CPU と主記憶 16GB を備えたサーバ上で行った．

評価用のテキストストリームとしては Twitter REST API[7]を用いて当研究室で収集したツイートから，東日本大震災発生当日である 2011 年 3 月 11 日の 12:00 以降のツイートを選んで用いた．係り受け解析の入力は文であるため，句点等を手がかりに文分割を行った．

本来であればテキストストリーム自体を入力とし，係り受け解析器の解析速度を計測することが望ましいが，今回用いた係り受け解析器では分類器による分類時間が大半の時間を占めること [9]，またより直接的に高速化の効果を計測するため，日本語係り受け解析器 J.DepP[8]を用いて上記のツイートを時系列順に係り受け解析し，実際に J.DepP が解いた係り受け解析の分類問題を時期列順で列挙して，これをストリームとみなして提案する分類器で分類し，その効果を確認することとした．

J.DepP は颯々野 [23] が提案した線形時間アルゴリズムに基づき，分類器を用いて与えられた二つの文節間の係り受け関係の有無を決定的に判定しながら文全体の係り受け構造を決定する．なるべく高精度の解析

[3] 低頻度の素性に関しては，その素性に関する全ての素性の重みを正負に分けて加算したものの方が正確な上限・下限となる場合があるので，両者を比較してより正確な上限・下限を利用する．

[4] トライを根ノードから順に辿ることで，同じ素性を二度チェックする必要がなくなるため．なお，ダブル配列から不要となった基本分類問題を削除する際は，その後すぐに別の基本分類問題を格納することを考慮し，削除によって空いた領域を詰め込み直すこと [18] は行わない．

[5] http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/cedar/

[6] 各素性インデクスを直前の素性番号との差分によって表現する．

[7] https://dev.twitter.com/docs/api

[8] http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/jdepp/

表 1: 評価に用いたテキストストリーム

	全ツイート	(公式 RT)
ツイート数	6,150,662	(1,556,038)
ツイート数/秒	142	(36)
文数	16,149,743	(6,072,658)
分類問題数	43,221,268	(20,383,651)

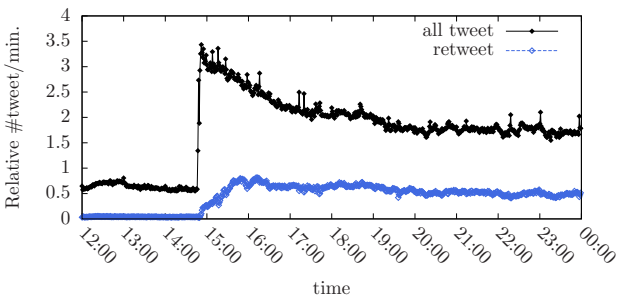

図 1: 東日本大震災当日のテキストストリームにおけるツイート数/分の変化

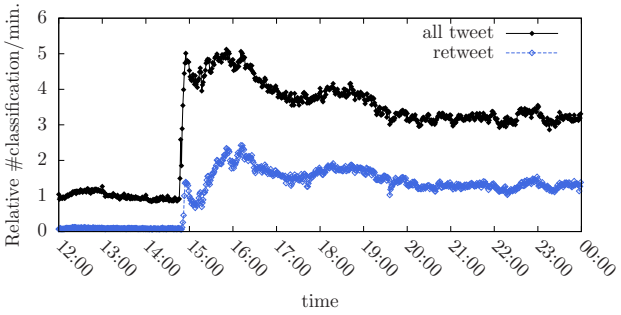

図 2: 東日本大震災当日のテキストストリームにおける分類問題数/分の変化

器である方が高速化の対象として望ましいため，3 次の多項式カーネルを用いて 3 つ組の素性の組み合わせまで考慮して分類器を学習した．得られた係り受け解析器を，京都大学テキストコーパス [24] Version 4.0[9]の標準分割[10]を用いて評価したところ，文節係り受け精度は 92.23% (文正解率 58.51%) であり，解析器のスタッキング [25] を行わない単独の係り受け解析器としては最高精度[11]となっている．

表 1 に評価に用いた東日本大震災発生当日のテキストストリームの詳細を示す．また，図 1, 2 に震災発生前の平均ツイート数および平均分類問題数を 1 とした

[9] http://nlp.ist.i.kyoto-u.ac.jp/index.php?京都大学テキストコーパス

[10] 学習データ: 一般記事 1 月 1，3〜11 日，社説 1〜8 月，計 24,263 文，234,685 文節，開発データ: 一般記事 1 月 12，13 日，社説 9 月，計 4,833 文，47,571 文節テストデータ: 一般記事 1 月 14〜17 日，社説 10〜12 月，計 9,284 文，89,874 文節．

[11] 現時点で最高精度とされるトーナメントモデル [26] では，京都大学コーパス Version 3.0 の標準分割で文節係り受け精度 91.96% (文正解率 57.44%) と報告されている．

表 2: 実験結果

手法	分類速度 [ミリ秒/分類問題]	メモリ [MiB]	速度向上比
ベースライン	0.0325	31.5	1.00
既存手法 [9]	0.0205	99.9	1.58
提案手法 (LFU)			
$k = 2^{10}$	0.0251	38.8	1.29
$k = 2^{12}$	0.0230	42.2	1.41
$k = 2^{14}$	0.0201	51.1	1.62
$k = 2^{16}$	0.0160	107.6	2.03
$k = 2^{18}$	0.0126	62.7	2.57
$k = 2^{20}$	0.0103	139.7	3.15
$k = 2^{22}$	0.0093	369.1	3.49
$k = 2^{24}$	0.0078	1287.8	4.18
$k = 2^{26}$	0.0074	3845.9	4.37
提案手法 (LRU)			
$k = 2^{10}$	0.0249	38.4	1.30
$k = 2^{12}$	0.0230	39.0	1.41
$k = 2^{14}$	0.0206	42.9	1.58
$k = 2^{16}$	0.0167	64.5	1.94
$k = 2^{18}$	0.0134	55.5	2.43
$k = 2^{20}$	0.0100	114.1	3.26
$k = 2^{22}$	0.0083	273.3	3.90
$k = 2^{24}$	0.0071	905.1	4.55
$k = 2^{26}$	0.0063	2309.5	**5.18**

1 分辺りのツイート数および分類問題数の時間変化を示す．収集したツイートは，API の制限のため，全体のツイートの一部をサンプルしたものであるが，それでも地震が発生した 14 時 46 分 18 秒を境として，災害によりツイート数が極端に増えていることが確認できる．また，ツイート数よりも，分類問題数の方が地震の発生前後で単位時間あたりの数の増加が大きいことは興味深い．今回用いた係り受け解析器の分類問題数は，文中の文節の数に比例するため，震災発生前後で個々のツイート中の文の長さが増加したものと推測される．

なお，全ツイート中で公式 RT（コピー）が占める割合は分類問題数で 47%程度であった，ツイート単位で解析結果をキャッシュするような単純なアプローチをとる場合には，オーバーヘッドを全て無視した場合でも，高々1.89(= 43221268/(43221268 − 20383651)) 倍程度の高速化に留まると考えられる．

表 2 に表 1 のテキストストリームを係り受け解析する際に生成された分類問題を，時系列順に処理したときの分類速度を示す．提案手法については，解析中に保持する基本分類問題数 k を 2^{10} から 2^{26} まで 2^2 倍刻みで変化させて分類速度を測定した．表中，ベースラインは式 3 を用いて分類したときの分類時間，手法 [9] は分類器を学習した新聞記事コーパス（脚注 10）から事前に基本分類を問題を静的に列挙し，式 4 で分類する分類器である．表から分かるように，提案手法を用いることでベースラインに対して，最大 5 倍の高速化を達成した．

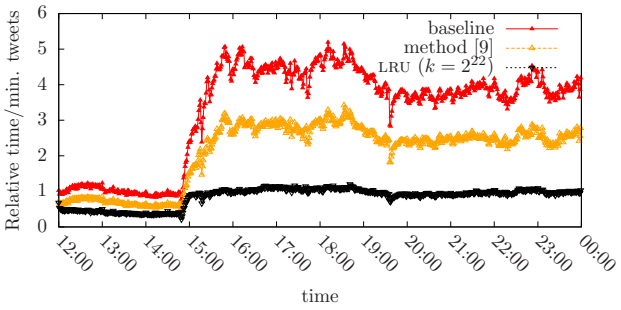

図 3: 東日本大震災当日のテキストストリームにおける 1 分間のツイートの処理に分類器が要する時間の変化

ここで，基本分類問題数を増やしていくと，基本分類問題を頻度 (LFU) ではなくアクセス時間 (LRU) に基づいて列挙する方が，高速化が顕著となることは興味深い．要因としては，時間的に近いツイート同士に類似した内容のテキストが多いためではないかと推察される．また，手法 [9] で列挙された基本分類問題の総数は 2,903,138 であるが，提案手法では $k = 2^{14}(= 16384)$ とこれと比べて大幅に少ない基本分類問題数で同程度の分類速度が得られていることは注目に値する．

図 3 に表 1 のテキストストリームを係り受け解析する際，1 分間のツイートを処理するのに分類器が要した分類時間の推移を示す．表中縦軸は，震災発生時刻までのベースライン手法の平均分類時間を 1 として分類時間を正規化したものである．ベースライン手法および既存手法 [9] では，分類問題の増加（図 2）に呼応して分類時間も増大しているが，提案手法では分類時間の増加は大幅に抑えられいる．このことから，提案手法はテキストストリームをリアルタイム解析をする上でより頑健であると結論づけることができる．

6 むすび

本稿では Twitter などのテキストストリームを頑健にリアルタイム解析することを目的として，言語解析で広く用いられる組み合わせ素性に基づく分類器をテキストストリームの内容に応じて適応的に高速化する手法を提案した．提案手法では，テキストストリームから基本分類問題を動的に列挙し，その分類結果を部分結果として保持して再利用することで，特にバースト時など特定の話題でテキストストリームの流量が増加した際に分類時間を削減する．

提案手法を係り受け解析に用いて東日本大震災発生当日のテキストストリームに対して適用したところ，分類速度を最大 5 倍高速化できることを確認した．

今後の課題としては，係り受け解析以外の様々な言語処理タスクの分類器の本手法を適用し，その効果を確認することがあげられる．組み合わせ素性に基づく線形分類器を用いた決定的な言語解析は，品詞分類や単語分割 [27]，構文解析 [28] といった多くの言語処理タスクで最高精度を達成していること，また従来構造学習が有効とされる固有表現認識などにおいても，構造翻訳 [29] (あるいは uptraining [30]) と呼ばれる手法で構造学習と同程度の精度を達成できるとの報告があることから，本研究の有効性が期待できる言語解析は，言語処理全般にわたると考えられる．

参考文献

[1] Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proceedings of ACL-HLT*, pp. 42–47, 2011.

[2] Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. #hardtoparse: POS tagging and parsing the Twitterverse. In *Proceedings of the AAAI-11 Workshop on Analyzing Microtext*, 2011.

[3] Bo Han and Timothy Baldwin. Lexical normalisation of short text messages: Makn sens a #twitter. In *Proceedings of ACL-HLT*, pp. 368–378, 2011.

[4] Ryohei Sasano, Sadao Kurohashi, and Manabu Okumura. A simple approach to unknown word processing in Japanese morphological analysis. In *Proceedings of IJCNLP*, pp. 162–170, 2013.

[5] Takeshi Sakaki, Fujio Toriumi, and Yutaka Matsuo. Tweet trend analysis in an emergency situation. In *Proceedings of the Special Workshop on Internet and Disasters*, 2011.

[6] Nobuhiro Kaji, Yasuhiro Fujiwara, Naoki Yoshinaga, and Masaru Kitsuregawa. Efficient staggered decoding for sequence labeling. In *Proceedings of ACL*, pp. 485–494, 2010.

[7] Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. Dual decomposition for parsing with non-projective head automata. In *Proceedings of EMNLP*, pp. 1288–1298, 2010.

[8] Alexander Rush and Slav Petrov. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of NAACL-HLT*, pp. 498–507, 2012.

[9] Naoki Yoshinaga and Masaru Kitsuregawa. Polynomial to linear: Efficient classification with conjunctive features. In *Proceedings of EMNLP*, pp. 1542–1551, 2009.

[10] Naoki Yoshinaga and Masaru Kitsuregawa. Efficient classification with conjunctive features. *Journal of Information Processing*, Vol. 20, No. 1, pp. 228–227.

[11] Vivek Srikumar, Gourab Kundu, and Dan Roth. On amortizing inference cost for structured prediction. In *Proceedings of EMNLP-CoNLL*, pp. 1114–1124, 2012.

[12] Gourab Kundu, Vivek Srikumar, and Dan Roth. Margin-based decomposed amortized inference. In *Proceedings of EMNLP*, pp. 905–913, 2013.

[13] Henning Wachsmuth, Benno Stein, and Gregor Engels. Constructing efficient information extraction pipelines. In *Proceedings of CIKM*, pp. 2237–2240, 2011.

[14] Henning Wachsmuth, Benno Stein, and Gregor Engels. Learning efficient information extraction on heterogeneous texts. In *Proceedings of IJCNLP*, pp. 534–542, 2013.

[15] Hideki Isozaki and Hideto Kazawa. Efficient support vector classifiers for named entity recognition. In *Proceedings of COLING 2002*, pp. 1–7, 2002.

[16] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of ICDT*, pp. 398–412, 2005.

[17] Jun ichi Aoe. An efficient digital search algorithm by using a double-array structure. *IEEE Transactions on Software Engineering*, Vol. 15, No. 9, pp. 1066–1077, 1989.

[18] Susumu Yata, Masaki Oono, Kazuhiro Morita, Masao Fuketa, and Jun ichi Aoe. An efficient deletion method for a minimal prefix double array. *Journal of Software: Practice and Experience*, Vol. 37, No. 5, pp. 523–534, 2007.

[19] 矢田晋, 田村雅浩, 森田和宏, 泓田正雄, 青江順一. ダブル配列による動的辞書の構成と評価. 第 71 回情報処理学会全国大会講演論文集, pp. 1263–1264, 2009.

[20] Nikolas Askitis and Ranjan Sinha. HAT-trie: A cache-conscious trie-based data structure for strings. In *Proceedings of the Thirtieth Australasian Conference on Computer Science*, pp. 97–105, 2007.

[21] Hugh E. Williams and Justin Zobel. Compressing integers for fast file access. *The Computer Journal*, Vol. 42, No. 3, pp. 193–201, 1999.

[22] Makoto Yasuhara, Toru Tanaka, Jun ya Norimatsu, and Mikio Yamamoto. An efficient language model using double-array structures. In *Proceedings of EMNLP*, pp. 222–232, 2013.

[23] 颯々野学. 日本語係り受け解析の線形時間アルゴリズム. 言語処理学会論文誌, Vol. 14, No. 1, pp. 3–16, 2007.

[24] 黒橋禎夫, 長尾眞. 京都大学テキストコーパス・プロジェクト. 言語処理学会第 3 回年次大会発表論文集, pp. 115–118, 1997.

[25] Masakazu Iwatate. *Development of Pairwise Comparison-based Japanese Dependency Parsers and Application to Corpus Annotation*. PhD thesis, 2012.

[26] 岩立将和, 浅原正幸, 松本裕治. トーナメントモデルを用いた日本語係り受け解析. 言語処理学会論文誌, Vol. 15, No. 5, pp. 169–185, 2008.

[27] Graham Neubig, Yosuke Nakata, and Shinsuke Mori. Pointwise prediction for robust, adaptable japanese morphological analysis. In *Proceedings of ACL*, pp. 529–533, 2011.

[28] Joakim Nivre and Ryan McDonald. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-HLT*, pp. 950–958, 2008.

[29] Percy Liang, Hal Daumé III, and Dan Klein. Structure compilation: trading structure for features. In *Proceedings of ICML*, pp. 592–599, 2008.

[30] Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. Uptraining for accurate deterministic question parsing. In *Proceedings of EMNLP*, pp. 705–713, 2010.