CRF に基づく形態素解析器のスコア計算の分割による モデルサイズと解析速度の調整

赤部晃一 [1]　神田峻介 [1]　小田悠介 [2]
[1]LegalOn Technologies Research [2] 東北大学 データ駆動科学・AI 教育研究センター
{koichi.akabe,shunsuke.kanda}@legalontech.jp
yusuke.oda.c1@tohoku.ac.jp

概要

CRF に基づく形態素解析器において，2-gram スコアの持ち方を変更し，用途に応じてモデルサイズと解析速度の調整を行うことを提案する．代表的な CRF に基づく形態素解析器では，CRF の 2-gram スコアを事前に計算して連接表に記憶しておくことで予測を効率的に行うが，素性の設計によってはスコアの事前計算は連接表を肥大化させる．本稿では，2-gram スコアの計算の一部を予測時に行うことで，連接表の肥大化を抑制する．また，2-gram スコアの計算を効率的に行うことで，予測時に追加的に生じる計算時間を抑制する．

1　はじめに

形態素解析は，日本語の自然言語処理における基礎技術であり，情報検索や機械学習の前処理として利用される．日本語における代表的な形態素解析器としては，MeCab [1] や KyTea [2] が挙げられる．このうち，MeCab は Conditional Random Fields (CRFs) [3] を利用してパラメータを学習し，入力文から形態素ラティスを生成し，単一のノードから得られる素性と，隣合う 2 つのノードに対応した素性を用い，各素性のパラメータを最尤推定する．

素性は解析精度等を考慮して設計されるが，設計によってはモデルサイズが肥大化する．公開されている学習済みモデルを比較すると，IPAdic [4] では連接表のサイズが $1,316 \times 1,316$ であるが，現代書き言葉 UniDic [5] (Ver. 3.1.0) は $15,626 \times 15,388$ であり，各スコアを 16-bit で表現すると，連接表だけで 459 MiB となる．UniDic では，素性テンプレートの見直しによってモデルサイズを軽量化する試みが行われているが，依然として組み込み機器や Web フロントエンド等の資源が限られた環境で利用するにはモデルサイズが大きく，可搬性が低い．

本稿では，まず CRF に基づく形態素解析について説明し，一部のスコア計算を解析時に行うことで，モデルのパラメータに手を加えずにモデルサイズを軽量化することを提案する．また，解析時のスコア計算を効率的に行う方法についても示す．

2　CRF に基づく形態素解析

2.1　定式化

CRF [3] はラティス上に定義される線形識別モデルであり，入力単語列を $\boldsymbol{x}$，ラベル列（品詞，読み等）を $\boldsymbol{y}$ とし，確率を式 (1) でモデル化する．

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp\left(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})\right)}{\sum_{\boldsymbol{y}} \exp\left(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})\right)} \tag{1}$$

ここで，$\boldsymbol{w}$ は重みベクトルであり，$\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ はラティス上の各パスに対応した素性ベクトルである．素性ベクトルを，各ノードに与えられる 1-gram 素性ベクトル ($\boldsymbol{\phi}_1$) と，各エッジに与えられる 2-gram 素性ベクトル ($\boldsymbol{\phi}_2$) に分ければ，式 (2) のように表すことが可能である．

$$\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) = \sum_i \boldsymbol{\phi}_1(\boldsymbol{x}, y_i) + \sum_{i,j} \boldsymbol{\phi}_2(\boldsymbol{x}, y_i, y_j) \tag{2}$$

ここで，y_i，y_j は隣り合う 2 つのノードを表す．1-gram 素性は辞書中の各エントリ e_i に付随した情報のみ利用しており，文脈に依存する情報を利用しない．すなわち，$\boldsymbol{\phi}_1(\boldsymbol{x}, y_i) := \boldsymbol{\phi}_1(e_i)$．このため，各エントリ e_i の 1-gram 素性のスコア $\boldsymbol{w}^\top \boldsymbol{\phi}_1(e_i)$ はエントリごとに事前に計算しておけば良い．一方，2-gram 素性には隣り合う 2 つのエントリの情報を用いる．すなわち，$\boldsymbol{\phi}_2(\boldsymbol{x}, y_i, y_j) := \boldsymbol{\phi}_2(e_i, e_j)$．単純に辞書内のすべてのエントリの組 (e_i, e_j) に対して $\boldsymbol{w}^\top \boldsymbol{\phi}_2(e_i, e_j)$ を事前計算することは，モデルを肥大化させて現実的でないため，実際には各エントリを

素性情報に基づいてグループ化し，グループのインデックスを用いて 2-gram 素性のスコアを参照する．

2.2 素性テンプレート

素性ベクトルの生成には複数の素性テンプレート関数が利用される．これは，辞書のエントリを引数とした関数であり，主に人手で設計される．素性テンプレート関数は 1-gram 素性と 2-gram 素性の両方に対して定義されるが，ここでは 2-gram 素性のみ着目する．K 個の素性テンプレートを用いると，2-gram 素性のスコアは式 (3) で表される．

$$\boldsymbol{w}^\top \boldsymbol{\phi}_2(e_i, e_j) = \sum_{k=1}^{K} \boldsymbol{w}^\top \boldsymbol{t}_k(e_i, e_j) \tag{3}$$

ここで，$\boldsymbol{t}_k$ は k 番目の 2-gram 素性テンプレート関数である．次に，式 (4) のように，左右のエントリ e_i, e_j から事前に素性テンプレートに対応した情報を取り出し，その組み合わせによって素性ベクトルを生成することを考える．

$$\boldsymbol{t}_k(e_i, e_j) = \boldsymbol{f}(l_k(e_i), r_k(e_j)) \tag{4}$$

ここで，l_k, r_k は，それぞれ左右のエントリから k 番目の素性テンプレートに対応した素性 ID を取得する関数，$\boldsymbol{f}$ は，左右の素性 ID の組に応じて素性ベクトルを返す関数である．

次に，各エントリから全ての素性テンプレートに対応した素性 ID を配列形式で取得する関数 $\boldsymbol{l}, \boldsymbol{r}$ を式 (5), (6) のように定義する．

$$\boldsymbol{l}(e_i) := (l_1(e_i), l_2(e_i), ..., l_K(e_i)) \tag{5}$$

$$\boldsymbol{r}(e_j) := (r_1(e_j), r_2(e_j), ..., r_K(e_j)) \tag{6}$$

さらに，左右の素性 ID の組 (l, r) に対応したスコア m を式 (7) のように事前に計算する．

$$m(l, r) := \boldsymbol{w}^\top \boldsymbol{f}(l, r) \tag{7}$$

これらを用い，各エントリの組に対応した 2-gram 素性のスコアを式 (8) で計算する．

$$\boldsymbol{w}^\top \boldsymbol{\phi}_2(e_i, e_j) = \sum_{k=1}^{K} m(l_k(e_i), r_k(e_j)) \tag{8}$$

3 2-gram スコアの部分的事前計算

3.1 連接表の大きさ

MeCab では，全ての $\boldsymbol{l}(e_i), \boldsymbol{r}(e_j)$ の組に対してスコアを事前に計算しておき，それらを連接表に格納する．重みの学習時に L1 正則化を行えばスパースなモデルが学習されるため，$\boldsymbol{l}(e_i), \boldsymbol{r}(e_j)$ は辞書中の複数のエントリで共通したものとなり，連接表のサイズをある程度抑えることが可能である．例えば現代書き言葉 UniDic では，辞書に登録されている語彙数は 1,879,222 語であるが，連接表のサイズは $15,626 \times 15,388$ であり，複数のエントリで共通した素性情報が参照されていることが確認できる．それでもなお，UniDic は素性テンプレートが詳細に設計されていることもあり，1 節でも述べたようにモデルが大きく可搬性が低い．

3.2 部分的事前計算

この問題への解決策として，全ての 2-gram スコアを事前に計算しておくのではなく，連接表肥大化の原因となる素性テンプレートの集合 S を予め定め，事前計算と解析時の計算に分割する．具体的には式 (9) でスコアを計算する．

$$\begin{aligned} \boldsymbol{w}^\top \boldsymbol{\phi}_2(e_i, e_j) &= \sum_{k \notin S} m(l_k(e_i), r_k(e_j)) \\ &+ \sum_{k \in S} m(l_k(e_i), r_k(e_j)) \end{aligned} \tag{9}$$

ここで，第 1 項は事前に計算する項，第 2 項は解析時に計算する項である．

S が大きいほど解析時に必要な計算が多くなるため，モデルサイズを効果的に軽量化できる小さい S を定める必要がある．本研究では S の選び方として以下の手法を比較する．

Various 生成される素性の種類が多い素性テンプレートを優先的に選択する．これは，品詞など種類数の少ない情報よりも，読みなど種類数の多い情報を含むほうがモデルが複雑化して連接表が大きくなりやすいのではないかという仮説に基づく．

Greedy 連接表のサイズに大きく影響を与える素性テンプレートを 1 つずつ調べ，貪欲的に選択していく．

Greedy-2 Greedy では，相関の強い素性テンプレートなど，同時に選択しなければ連接表が軽量化しない素性テンプレートが選択されない．そこで，2 つの素性テンプレートを同時に選択した際の連接表のサイズ変化も考慮する．

4 ダブル配列を用いたスコア辞書

2-gram スコアを解析時に計算すると，解析速度が低下すると考えられる．そこで，式 (9) の計算を効

Algorithm 1 ダブル配列を用いたスコア計算

Input:

Double array: BASE, CHECK, VALUE

Word entries: e_i, e_j, Index set: S,

Feature template functions: $\boldsymbol{l}, \boldsymbol{r}$

Output: Score: s

$s \longleftarrow 0$
for $k \in S$ **do**
  $k_1 \longleftarrow l_k(e_i)$
  $k_2 \longleftarrow r_k(e_j)$
  $t \longleftarrow \mathsf{BASE}[k_1] \oplus k_2$
  if $\mathsf{CHECK}[t] = k_1$ **then**
    $s \longleftarrow s + \mathsf{VALUE}[t]$
  end if
end for
return s

率的に行うために，関数 m の引数から値へのマッピングにダブル配列 [6] を用いることを提案する．

ダブル配列はトライ [7] の効率的な実装であり，2 つの配列 BASE と CHECK，及び値配列 VALUE によって構成される key-value マップである．整数値 k_1, k_2, t について式 (10) が満たされるとき，キー (k_1, k_2) の値が $\mathsf{VALUE}[t]$ であると見なす．

$$\mathsf{BASE}[k_1] \oplus k_2 = t \wedge \mathsf{CHECK}[t] = k_1 \qquad (10)$$

ここで演算子 $\oplus$ は排他的論理和を表す[1)]．ダブル配列を用いると，スコア計算は Algorithm 1 のとおりとなる．

ダブル配列は，1) 配列の要素の取得，2) 排他的論理和の計算，3) 比較，という単純な処理によって構成されるため，Algorithm 1 内のループ処理は SIMD 命令[2)]によって容易に並列化可能である．

5 実験

5.1 実験設定

モデル学習のためのデータセットとして，現代日本語書き言葉均衡コーパス (BCCWJ [8], Ver. 1.1) のうち，人手で短単位アノテーションされたコアデータ 60k 文を利用した．辞書には現代書き言葉 UniDic [5] (Ver. 3.1.0) を利用し，各パラメータを再学習した．

1) ダブル配列の定義によっては排他的論理和ではなく和 (+) が用いられることもある．
2) メモリ上の連続していないデータを読み込むため，AVX2 の Gather 命令を利用する必要がある．

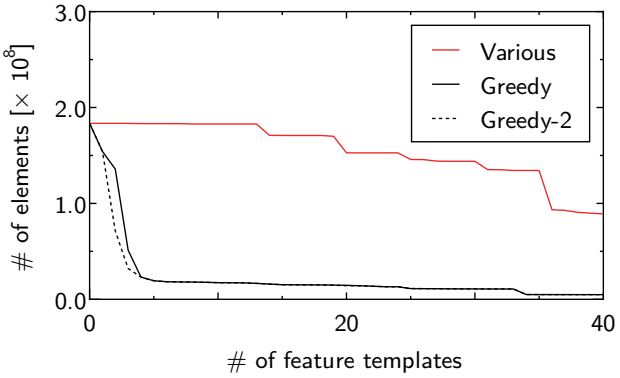

図 1 2-gram スコアの事前計算から除外する素性テンプレートの数と連接表の要素数の関係

ベースラインとして CRF に基づく形態素解析器である Vibrato[3)] を利用し，提案法は Vibrato を修正する形で実装した．CRF の学習には rucrf[4)] を用い，OWL-QN 法 [9] による L1 正則化 [10] を行った．

計算を効率的に行うため，重み，及び統合後のスコアはすべて量子化した．スコアの事前計算では量子化前の値が用いられるが，解析時の計算では量子化後の値が用いられるため，手法によって算出されるスコアが僅かに異なる．しかし，その差は解析結果に大きな影響を与えるものではない．

実装にはプログラミング言語 Rust を用いた．コンパイラは rustc 1.65，コンパイルオプションは `-C opt-level=3` とした．

解析速度の測定には，BCCWJ のコアデータ以外からランダムに抽出した 100k 文を利用し，100 回解析した平均値を用いた．測定の際は，分割対象の文を先に RAM に読み込み，IO 処理にかかる時間は除外した．

実験は以下のスペックのマシンを用いてシングルスレッドで実施した．

- CPU: Intel Core i9-12900K CPU @ 3.2GHz
- Memory: 64GiB RAM (L1: 640KiB, L2: 14MiB)

5.2 スコアの部分的事前計算の効果

5.2.1 連接表のサイズの変化

解析時に統合する素性テンプレートのインデックス集合 S のサイズを変化させた際の連接表の要素数の変化を図 1 に示す．

図を見ると，Greedy または Greedy-2 によって優先的に選択される素性テンプレートのうち，最初の

3) https://github.com/daac-tools/vibrato
4) https://github.com/daac-tools/rucrf

表 1 2-gram モデルサイズ [MiB] と解析時間 [s] の関係

手法	サイズ	時間
Matrix	350.0	0.87
Dual	41.1	2.13
Raw	1.9	20.04

5 つの素性テンプレートをスコアの事前計算の対象から除外するだけで，連接表のサイズが 10%程度まで軽量化することが確認された．一方で，Various によって優先的に選択される素性テンプレートを除外しても，連接表の軽量化効果が殆どないことが確認された．

この結果から，Greedy または Greedy-2 によって貪欲的に選択される素性テンプレートを除外することが，連接表の軽量化に有効であることが分かる．

Greedy-2 については，初期の軽量化効果は Greedy と比較して大きいものの，すぐに Greedy と大差が無くなることが確認された．これは，2 つの素性テンプレートのみが相関しているケースが少なかったためと考えられる．他の素性テンプレートを除外した場合の影響を考慮する場合，計算量は考慮する素性テンプレートの数 n に対して指数関数的に増大するため，n を増やすことは現実的ではない．このため，以降は単純に大きな軽量化効果を得られる Greedy を採用した．

5.2.2 モデルサイズと解析時間の比較

2-gram スコアの事前計算から除外する素性テンプレートの数を変化させた際の，2-gram モデルサイズと解析時間への影響を調査するため，以下の 3 つの状況について測定した．ここにおける 2-gram モデルサイズとは，連接表のサイズと 2-gram スコアを格納したダブル配列のサイズの総和である．

Matrix 全ての 2-gram スコアを事前に計算して連接表に格納する．(ベースライン)

Dual Greedy によって優先的に選択される 8 個の素性テンプレートについて連接表から除外し，解析時に計算を行う．

Raw 全ての 2-gram スコアを解析時に計算し，連接表は利用しない．($S = \{1, 2, ..., K\}$)

これらの結果を表 1 に示す．Dual では，Matrix に比べて 2-gram モデルサイズが 12%以下になっていることを確認できる．スコアの事前計算から除外する素性テンプレートを増やすと，必然的にダブル配列のサイズは大きくなるが，その増加量は連接表の軽量化した分に比べれば十分に小さく，モデル全体の軽量化に効果的であることが分かる．一方で，Dual の解析時間は Matrix の 2.4 倍に増加しており，解析時に 2-gram スコアを計算することで解析が遅くなることが伺える．

ここからモデルサイズと速度はトレードオフの関係であり，インデックス集合 S のサイズは，許容されるモデルサイズと解析速度によって調整する必要があることが分かる．

Raw は Dual と比較して傾向が顕著であり，2-gram モデルサイズは Matrix の 0.5%である一方，解析時間は Matrix の 23 倍であった．

表 2 SIMD による解析時間 [s] の変化

手法	SIMD なし	SIMD あり
Dual	2.13	1.80
Raw	20.04	12.05

5.3 スコア計算に SIMD を利用する効果

ここでは，解析時の 2-gram スコアの計算を SIMD 並列化することの効果を調査する．5.2.2 節の Dual および Raw の設定について，Algorithm 1 の処理を 8 並列化[5] した場合の解析時間の変化を表 2 に示す．

この結果から，ダブル配列を用いたスコア計算が SIMD 命令によって実際に高速化されたことを確認できる．ただし，全ての 2-gram スコアを事前計算して参照する Matrix と比較すると，Dual は SIMD 命令を用いても 2 倍以上遅いことが確認できる．

6 まとめ

本稿では，CRF に基づく形態素解析器において，2-gram スコアの計算を事前計算と解析時の計算に分割することで，モデルサイズを軽量化する手法を提案した．また，解析時の計算を SIMD 命令で高速化できることを示した．提案法は，パラメータに手を加えるなど，解析精度に影響を与えることは行わない．このため，解析精度を維持したまま，組み込み機器や Web フロントエンドなど，資源が限られた環境でも利用可能なモデルを配布することが可能となる．

今回の実験では，事前計算から除外する素性テンプレートを選択する際に，そのテンプレートの内容までは考慮しなかったが，言語学的な知見を取り入れることによる効果についても検討の余地がある．

5) AVX2 で 32-bit 演算を並列化できる最大の個数が 8 である．

参考文献

[1] Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. Applying conditional random fields to Japanese morphological analysis. In **Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing**, pp. 230–237, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[2] Graham Neubig, Yosuke Nakata, and Shinsuke Mori. Pointwise prediction for robust, adaptable Japanese morphological analysis. In **Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies**, pp. 529–533, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[3] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In **Proceedings of the Eighteenth International Conference on Machine Learning**, ICML '01, p. 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[4] Masayuki Asahara and Yuji Matsumoto. ipadic version 2.7.0 User's Manual. https://ja.osdn.net/projects/ipadic/docs/ipadic-2.7.0-manual-en.pdf, 2003.

[5] 岡照晃. CRF 素性テンプレートの見直しによるモデルサイズを軽量化した解析用 UniDic — unidic-cwj-2.2.0 と unidic-csj-2.2.0 —. 言語資源活用ワークショップ 2017 発表予稿集, 143–152.

[6] 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.

[7] Edward Fredkin. Trie memory. **Commun. ACM**, Vol. 3, No. 9, p. 490–499, sep 1960.

[8] 前川喜久雄. 代表性を有する大規模日本語書き言葉コーパスの構築 (〈特集〉日本語コーパス). 人工知能, Vol. 24, No. 5, pp. 616–622, 2009.

[9] Galen Andrew and Jianfeng Gao. Scalable training of l1-regularized log-linear models. In **Proceedings of the 24th International Conference on Machine Learning**, ICML '07, p. 33–40, New York, NY, USA, 2007. Association for Computing Machinery.

[10] Robert Tibshirani. Regression shrinkage and selection via the lasso. **Journal of the Royal Statistical Society: Series B (Methodological)**, Vol. 58, No. 1, pp. 267–288, 1996.