Vaporetto: 点予測法に基づく高速な日本語トークナイザ

赤部 晃一 [1]　神田 峻介 [1]　小田 悠介 [1,2]　森 信介 [3]
[1] LegalForce Research　[2] 東北大学 データ駆動科学・AI 教育研究センター
[3] 京都大学 学術情報メディアセンター
{koichi.akabe, shunsuke.kanda, yusuke.oda}@legalforce.co.jp
forest@i.kyoto-u.ac.jp

概要

線形分類モデルによる点予測法に基づくトークナイザに関し，高速な計算手法を開発し，その実装である Vaporetto[1)] について報告する．本手法では，予測の際に圧縮ダブル配列を用いたオートマトンを採用し，高速な文字列探索を実現する．また，スコアに対して 3 種類の前処理を提案する．実験の結果，既存の点予測法に基づくトークナイザである KyTea と比較し，5.4 倍の高速化が確認された．また，提案した前処理手法については，その全てが高速化に寄与することを確認した．

1　はじめに

日本語や中国語など，文中に単語境界を示す区切り記号を挿入しない言語では，自然言語処理の前処理としてトークン化を行う必要がある．これには大きく分けて 2 種類の手法が利用される．1 つは系列予測法と呼ばれ，入力文から辞書に基づくラティスを構築し，語や語間に対して与えられるコストが最小化される経路を探索する．代表的なものでは MeCab [1] が採用している．もう 1 つは点予測法と呼ばれ，入力文のすべての文字間に対し，語の境界であるか否かを識別モデルによって判別する．代表的なものでは KyTea [2] が採用している．

トークン化は前処理として広く利用されており，大量のテキストを処理するとき，分割の精度だけでなく実行速度も重要である．本研究では，既存の点予測法に基づくトークン化を，モデルに手を加えずに高速に動作させる手法を提案する．

2　点予測法によるトークン化

点予測法では，入力文の各文字境界に対して指定された幅の窓を考え，ここに含まれる文字列から識別モデルの素性を生成する [3]．KyTea では素性として，文字 *n*-gram，文字種 *n*-gram，辞書単語素性の 3 種類を利用して線形分類モデルを学習する．

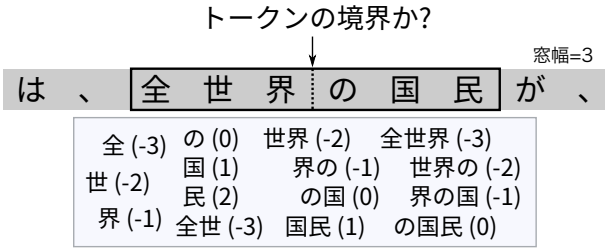

図 1　点予測法に基づくトークン化の例．括弧内の数字は，判定箇所に対する各 *n*-gram の相対位置を示す．

図 1 は点予測法によるトークン化の例である．この例では，入力文として「は、全世界の国民が、」が与えられ，「界」と「の」の間がトークン境界であるか否かを文字 *n*-gram 素性を用いて識別している．文字 *n*-gram 素性は，窓に含まれる文字 *n*-gram と，その相対位置の組で表現される．文字種 *n*-gram についても素性の作り方は文字 *n*-gram の場合と同様だが，各文字をひらがな，カタカナ，漢字，数字，ローマ字，その他の 6 種類に分類し，文字の代わりに文字種記号の配列を利用する．図 2 は辞書単語素性の例である．この例では，単語辞書に「の」と「世界」が含まれる場合に，太線で示された文字境界に対してどのような素性が生成されるかを示している．KyTea と MeCab の精度面による比較分析は，先行研究 [4] で行われている．

3　アルゴリズムの高速化

2 節で示したトークン化手法は，入力文に対する 3 種類のパターン集合を用いたオーバーラップパターンマッチと，マッチ箇所に対するスコアの加算という 2 つの問題に帰着される．

図 3 は，入力文の各文字境界に対してスコアを計

1）https://github.com/legalforce-research/vaporetto

図 2　辞書単語素性の例．3-gram 以上の場合は，複数の文字境界について「内側」素性を生成する．

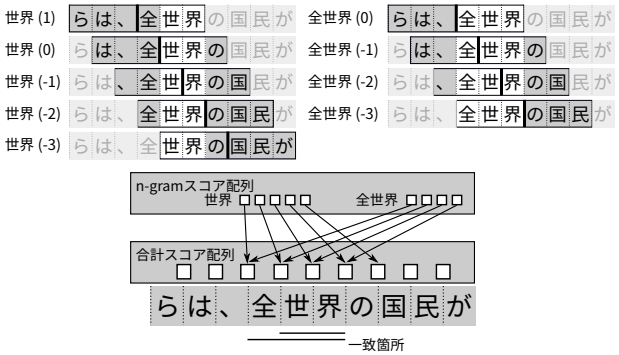

図 3　n-gram スコア配列の合計スコア配列への加算の例．ここでは 2 つの n-gram に対して括弧内で示された相対位置に応じたスコアが学習されており，スコア計算の際は各 n-gram スコアが合計スコア配列に加算される．

算する例である．窓幅を W とすると，各 n-gram について文字境界との相対位置に応じた $2W - n + 1$ 通りのスコアが学習される．トークン境界の予測時には，各 n-gram をパターンとして，入力文に対するパターンマッチを行い，マッチしたパターンとその位置から，どの文字境界にどのスコアを加算するかを決定する．

本節では，パターンマッチとスコア計算のそれぞれの部分について，処理の高速化手法を紹介する．

3.1　パターンマッチの高速化

MeCab に代表される系列予測法に基づくトークナイザの多くは，パターンマッチにトライ法 [5] を採用している．トライ法によるパターンマッチは，入力文の長さを N，パターンの平均長を K としたとき $O(NK)$ 時間で動作する．

一方，提案法では Aho-Corasick (AC) 法 [6] によるパターンマッチオートマトン (PMA) を採用する．AC 法は，マッチする解の個数を occ とすると $O(N + occ)$ 時間で動作し，3.2 節で提案する高速化手法とも親和性がある．また，PMA を圧縮ダブル配列 [7] で実装することで高速な解析を実現する．

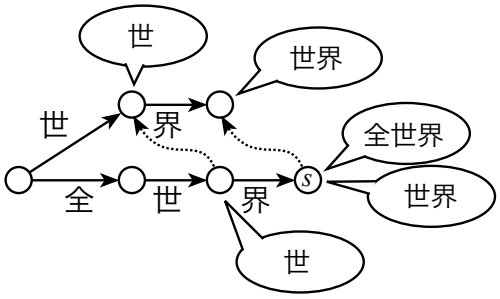

図 4　「世」「世界」「全世界」を登録した PMA．状態 s は「全世界」と「世界」を報告する出力状態．点線は Failure リンクを示しており，ルートへの Failure リンクは省略している．

3.2　n-gram スコアの事前加算

PMA に登録されているパターン集合を $\mathcal{P}$ とし，あるパターン $p \in \mathcal{P}$ の n-gram スコア配列を $\boldsymbol{w}_{\text{pattern}}(p)$ とする．AC 法による PMA は，出力状態 s に到達した際にマッチしたパターン集合 $\mathcal{S}(s) \subseteq \mathcal{P}$ を報告する．点予測法では PMA が報告した $\mathcal{S}(s)$ について，全ての $q \in \mathcal{S}(s)$ の n-gram スコア配列 $\boldsymbol{w}_{\text{pattern}}(q)$ を合計スコア配列に加算することで文字境界のスコアを算出する．例えば図 3 は，$\mathcal{S}(s)$ = { 世界, 全世界 } について，n-gram スコア配列 $\boldsymbol{w}_{\text{pattern}}$(世界) と $\boldsymbol{w}_{\text{pattern}}$(全世界) を合計スコア配列に加算する様子を示している．

提案法では，各 q の n-gram スコア配列を別々に加算するのではなく，まとめて加算することで合計スコアの計算を高速化する．具体的には，各出力状態 s について以下のスコア配列 $\boldsymbol{w}_{\text{state}}(s)$ を定義する．

$$\boldsymbol{w}_{\text{state}}(s) := \sum_{q \in \mathcal{S}(s)} \boldsymbol{w}_{\text{pattern}}(q)$$

PMA が出力状態 s に到達し $\mathcal{S}(s)$ が報告された際は，$\boldsymbol{w}_{\text{pattern}}(q)$ を参照する代わりに $\boldsymbol{w}_{\text{state}}(s)$ を参照する．

図 4 は，「世」「世界」「全世界」という 3 つのパターンを登録した PMA を示している．出力状態 s は $\mathcal{S}(s)$ = { 世界, 全世界 } を報告する．提案法では，これらの n-gram スコア配列を事前に加算し，配列 $\boldsymbol{w}_{\text{state}}(s)$ を状態 s に持たせておくことで合計スコア配列への加算回数が低減される．

3.3　辞書単語素性のスコアの事前加算

辞書中の単語のうち，文字 n-gram 素性の文字列と厳密一致する単語については辞書から削除し，辞書単語素性のスコアを事前に文字 n-gram 素性のスコアに加算しておく．これにより，辞書単語素性 PMA からのパターン報告回数が減少し，文字 n-gram 素

性のスコアと辞書単語素性のスコアをまとめて合計スコア配列に加算することが可能となる．

3.4 文字種 n-gram スコアの事前加算

文字種 n-gram によって加算される文字境界のスコアは，窓によって切り出される文字列のみによって一意に決まる．そのため，長さ $2W$ の系列の全組合せについて事前に計算したスコアを保存しておけば，解析時のパターンマッチとスコアの加算を回避できる．文字種は高々 6 種類のため，各文字種に 3 ビットのコード値を割り当てれば，文字種系列は $6W$ ビットの整数値で表現できる．この整数値を系列の ID とし，計算済みスコアへの参照に用いる．$W = 3$ のとき，系列の ID は高々 262,144 通りであり，各スコアを 4 バイトで保持するとメモリ使用量は 1MiB となる．

入力文として文字種列 $(a_0, a_1, \ldots, a_{N-1})$ が与えられたとき，各文字境界についての系列の ID は以下のように表現できる．文字 a_{i-1} と a_i の境界に対応した系列 $A_i = (a_{i-W}, a_{i-W+1}, \ldots, a_{i+W-1})$ の ID を $\mathtt{Id}(A_i)$ と表記する．$i \notin [0, N)$ な a_i は範囲外を表現する特殊文字として扱い，そのコード値は 0 とする．このとき，$\mathtt{Id}(A_i)$ は以下の漸化式で記述できる．

$$\mathtt{Id}(A_{i+1}) = ((\mathtt{Id}(A_i) \ll 3) \mid \mathtt{Cd}(a_{i+W})) \,\&\, (2^{6W} - 1)$$

ただし，初項は $\mathtt{Id}(A_{-W}) = 0$ である．式中の $\mathtt{Cd}(a) \in [0, 8)$ は文字種 a のコード値，“«” は左シフト演算，“|” は OR 演算，“&” は AND 演算である．

漸化式が示すように，$\mathtt{Id}(A_i)$ を用いて $\mathtt{Id}(A_{i+1})$ が計算できる．すなわち $\mathtt{Id}(A_{-W}) = 0$ を起点とし，文字境界を 1 文字ずつ右に移動しながら，各系列の ID がビット演算で高速に計算できる．図 5 は文字種系列の ID をビット演算を用いて計算する例である．

4 L1 正則化によるモデルの圧縮

L1 正則化 [8] を利用すれば重みベクトルがスパースとなり，過学習を防ぎつつ小さなモデルを学習することが可能である．線形分類モデルの学習で L1 正則化を行うには，以下の目的関数を最小化する重みベクトル $\boldsymbol{w}$ を学習する．

$$\min_{\boldsymbol{w}} \|\boldsymbol{w}\|_1 + C \sum_i \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$$

ここで $\xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$ は i 番目の入力 $\boldsymbol{x}_i$ と出力 y_i から計算される損失関数，C は罰則パラメータである[2)]．

2) 通常，罰則パラメータ C は罰則項の係数とするが，LIBLINEAR では目的関数の係数としている．ここではLIBLINEAR の定式化に合わせている．

図 5 文字種系列の ID を計算する例．窓幅 $W = 3$ としている．文字種の下の数字は，各文字種に対応したコード値を表す．$\mathtt{Cd}(\mathrm{H}) = 001_{(2)}$ はひらがな，$\mathtt{Cd}(\mathrm{K}) = 101_{(2)}$ は漢字，$\mathtt{Cd}(\mathrm{O}) = 110_{(2)}$ はその他の文字種に対応している．次の系列 ID は，現在の系列 ID に対するビット演算によって求まる．

C が小さいほど罰則項の比重が大きくなり，正則化をより強くかけることとなる．モデルサイズを小さくすると，合計スコアの計算に必要な各スコアの加算回数が減ることに加え，CPU キャッシュの利用頻度も上がり，処理の高速化が期待される．本研究では，モデルのサイズと分割速度の関係を調査し，モデルサイズの調整による高速化も検討する．

5 実験

提案法の高速化効果を検証するため，提案法と各種ツールの分割速度の比較を行った．また，正則化係数に応じた精度の比較も行った．

5.1 実験設定

点予測法の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 [9], 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 60,004 文を利用した．また，辞書として UniDic [10] (Ver. 3.1.0) のうち半角空白記号を含まない 879,048 語を利用した．各実験では，BCCWJ のコアデータ 60,004 文を 10 分割して交差検証を行った．速度の測定では，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．モデルの学習には LIBLINEAR [11] の L1-regularized L2-loss SVC を利用し，$C = 1.0$ とした．また，窓幅 $W = 3$ とし，n-gram の最大長は 3 とした．

3 節で提案した各手法の効果を確認するため，各手法を個別に適用した場合とすべて適用した場合の

表 1　分割速度の比較．単位は [100 万文字/秒]．罰則パラメータ C は 1.0 とした．

ツール/設定	速度	σ
KyTea (2020-04-03)	1.390	0.037
Vaporetto (Ours)		
(a) 圧縮ダブル配列の利用 (3.1 節)	5.814	0.110
(b) a + n-gram スコア事前加算 (3.2 節)	6.556	0.102
(c) a + 辞書単語スコア事前加算 (3.3 節)	6.473	0.107
(d) a + 文字種スコア事前加算 (3.4 節)	6.372	0.140
(e) a + b + c + d	**7.544**	0.101
MeCab (2020-09-14)	3.848	0.110
sudachi.rs (0.6.2)	0.821	0.013

速度を測定した．比較対象として，既存の点予測法に基づくトークナイザである KyTea の分割速度を測定した．[3)]

また，学習データやモデルが異なるため単純な比較はできないが，MeCab と sudachi.rs[4)] によってトークン化した際の実行速度も測定した．MeCab では IPADIC を利用し，sudachi.rs では SudachiDict-core (Ver. 20210802) による短単位相当の分割を行った．

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(R) Core(TM) i7-8086K CPU @ 4.00GHz
- Memory: 64GiB RAM (L1: 32KiB, L2: 256KiB)

5.2　実験結果

5.2.1　分割速度の比較

分割速度の比較結果を表 1 に示す．実験結果によると，提案法では素性のスコアの前処理をしない場合でも KyTea の 4 倍以上の速度でトークン化処理を行うことが可能であり，圧縮ダブル配列を利用した AC 法の効果が大きいことが分かる．また，(b)〜(d) の各手法を単独で実施した場合と同時に実施した場合のいずれの場合も処理が高速化された事が確認された．

5.2.2　L1 正則化，分割精度，分割速度の関係

図 6 は，罰則パラメータ C を変化させた際の，分割誤り率と分割速度の関係を示している．C が小さい領域では，精度と速度はトレード・オフの関係にあり，正則化を強くするほど精度が下がる一方，分割速度は速くなる．しかしながら，C が大きい領域では正則化が弱まるにつれて分割速度が遅くなるだけでなく，過学習によって分割誤りも増えている．

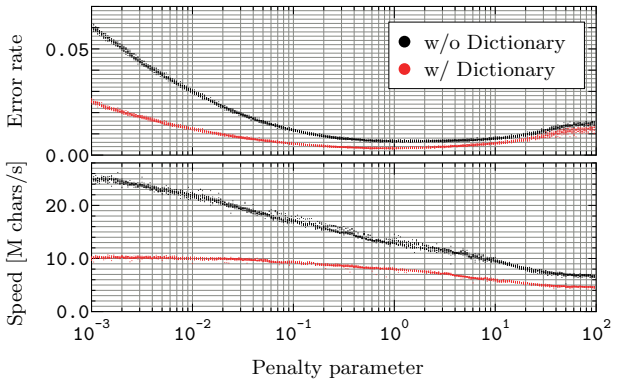

図 6　罰則パラメータを変化させた際の，分割誤り率と分割速度の関係．

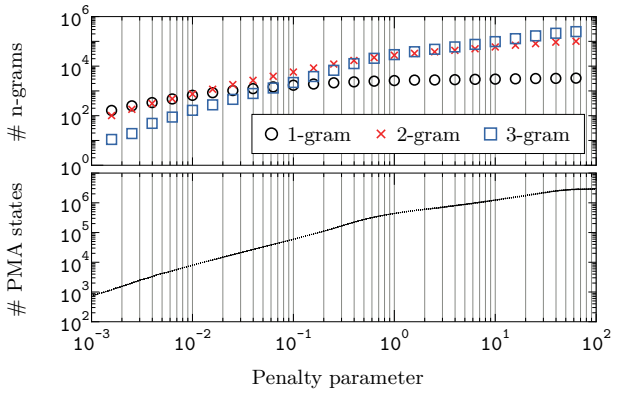

図 7　罰則パラメータを変化させた際の，スコアが 0 でない n-gram の数と PMA の状態数の関係．

次に，図 7 は C を変化させた際の，スコアが 0 でない n-gram の個数と PMA の状態数の関係を示している．正則化を強くするほど，n-gram がモデルから削除され PMA が小さくなるが，その際に長い n-gram が優先的にモデルから削除され，短い n-gram が支配的になることが確認できる．

6　まとめ

本研究では，線形分類モデルを用いた点予測法に基づく日本語のトークン化を，パターンマッチとスコアの加算という 2 つの部分問題に帰着させ，各部分に対して最適化を行った．

提案したアルゴリズムは日本語のトークン化に特化したものではなく，AC オートマトンを利用した点予測法に一般化することが可能なため，他分野への応用も検討したい．

3)　KyTea と提案法ではスコアを量子化しており，スコアの加算は整数の演算によって行われる．分割精度は，KyTea と提案法の実装の違いによる軽微な差を除き同じである．

4)　sudachi.rs は Sudachi[12] の高速な実装である．https://github.com/WorksApplications/sudachi.rs

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] Manabu Sassano. An empirical study of active learning with support vector machines for Japanese word segmentation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 505–512, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[4] 森信介, 中田陽介, Neubig Graham, 河原達也. 点予測による形態素解析. 自然言語処理, Vol. 18, No. 4, pp. 367–381, 2011.

[5] Edward Fredkin. Trie memory. *Communications of the ACM*, Vol. 3, No. 9, pp. 490–499, 1960.

[6] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, Vol. 18, No. 6, pp. 333–340, June 1975.

[7] Susumu Yata, Masaki Oono, Kazuhiro Morita, Masao Fuketa, Toru Sumitomo, and Jun-ichi Aoe. A compact static double-array keeping character codes. *Inf. Process. Manage.*, Vol. 43, No. 1, p. 237–247, January 2007.

[8] 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.

[9] 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.

[10] 伝康晴, 小木曽智信, 小椋秀樹, 山田篤, 峯松信明, 内元清貴, 小磯花絵. コーパス日本語学のための言語資源 : 形態素解析用電子化辞書の開発とその応用. 日本語科学, Vol. 22, pp. 101–123, oct 2007.

[11] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, Vol. 9, p. 1871–1874, June 2008.

[12] Kazuma Takaoka, Sorami Hisamoto, Noriko Kawahara, Miho Sakamoto, Yoshitaka Uchida, and Yuji Matsumoto. Sudachi: a Japanese tokenizer for business. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

A　データや分割単位の違いによる速度の比較

表 2 は，分割単位を変更した際の各ツールの速度を比較したものである．KyTea と Vaporetto では，学習コーパスとして BCCWJ の短単位 (SUW) と長単位 (LUW) のいずれかを利用し，単語辞書は利用していない．MeCab では，SUW として UniDic，LUW として IPADIC を利用した．Sudachi では SUW として UniDic 相当，LUW として固有表現 (NE) 相当のオプションを利用した．

表 2　分割速度の比較．単位は [100 万文字/秒].

ツール/設定	速度	σ
KyTea / BCCWJ-SUW	1.485	0.043
KyTea / BCCWJ-LUW	1.520	0.042
Vaporetto / BCCWJ-SUW	12.595	0.325
Vaporetto / BCCWJ-LUW	12.698	0.385
MeCab / Unidic	1.426	0.026
MeCab / IPADIC	3.848	0.110
sudachi.rs / SUW	0.821	0.013
sudachi.rs / NE	0.865	0.012

KyTea，Vaporetto，Sudachi，sudachi.rs では，分割単位の違いによる速度の大きな変化は確認できない．一方，MeCab では UniDic を使用した場合に IPADIC に比べて 2 倍程度の時間を要していることが確認できる．MeCab で BCCWJ を解析した際の L1 キャッシュミスの回数を `perf` コマンドで測定したところ，IPADIC を使用した際は 3.6×10^6 回であったのに対し，UniDic を使用した際は 63.1×10^6 回であった．UniDic を使用した場合，キャッシュミスが多く発生していることが速度低下の原因となっていることが伺える．

B　L1 キャッシュミスの頻度

図 8 は，罰則パラメータ C を変化させた際の L1 キャッシュミスが発生する回数を示している．BCCWJ の 60,004 文で 10 交差検証を行った際の，各セットでの合計値を示した．キャッシュミスの回数は `perf` コマンドで測定した．

単語辞書を利用しない場合，$C < 0.1$ まではモデルサイズが小さいため，キャッシュミスの回数が 1 万回程度で抑えられており，キャッシュが効率的に利用されていることが伺える．しかし，$C > 0.1$ では次第にモデルがキャッシュに乗り切らなくなり，キャッシュミスの回数が増えていくことが確認できる．

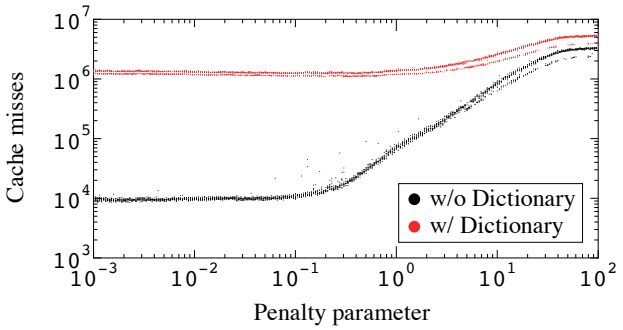

図 8　罰則パラメータを変化させた際の，L1 キャッシュミス回数の変化．

また，単語辞書を利用する場合は，C が小さい領域でもキャッシュミスが多く発生していることが確認できる．

C　窓幅と n-gram サイズを変化させた際の分割速度と分割精度の関係

表 3 は窓幅 W と n-gram サイズを変化させた際の，分割速度と分割精度の関係を示している．学習には BCCWJ のみ利用し，単語辞書は利用していない．$W \geq 4$ の場合，文字種 n-gram のスコアを事前計算するとメモリを多く消費するため，文字 n-gram と同様に PMA を利用している．

表 3　窓幅ごとの分割速度と分割精度．速度の単位は [100 万文字/秒].

W, n	SUW			LUW		
	速度	F_1	誤り率	速度	F_1	誤り率
1,1	27.20	0.9294	0.0883	27.86	0.9230	0.0750
2,2	15.68	0.9926	0.0093	16.26	0.9840	0.0155
3,3	12.60	0.9947	0.0066	12.70	0.9910	0.0088
4,4	10.84	0.9947	0.0066	10.76	0.9919	0.0078
5,5	9.43	0.9945	0.0068	8.93	0.9920	0.0077
6,6	8.28	0.9943	0.0071	7.73	0.9919	0.0078
7,7	7.74	0.9941	0.0074	7.15	0.9918	0.0080

SUW の場合，$W = n = 3$ で精度が上限となり，LUW の場合は $W = n = 5$ で精度が上限となっていることが確認できる．