Kernel Slicing: Scalable Online Training with Conjunctive Features

Naoki Yoshinaga
Institute of Industrial Science,
the University of Tokyo
ynaga@tkl.iis.u-tokyo.ac.jp

Masaru Kitsuregawa
Institute of Industrial Science,
the University of Tokyo
kitsure@tkl.iis.u-tokyo.ac.jp

Abstract

This paper proposes an efficient online method that trains a classifier with many conjunctive features. We employ kernel computation called *kernel slicing*, which explicitly considers conjunctions among frequent features in computing the polynomial kernel, to combine the merits of linear and kernel-based training. To improve the scalability of this training, we reuse the temporal margins of partial feature vectors and terminate unnecessary margin computations. Experiments on dependency parsing and hyponymy-relation extraction demonstrated that our method could train a classifier orders of magnitude faster than kernel-based online learning, while retaining its space efficiency.

1 Introduction

The past twenty years have witnessed a growing use of machine-learning classifiers in the field of NLP. Since the classification target of complex NLP tasks (*e.g.*, dependency parsing and relation extraction) consists of more than one constituent (*e.g.*, a head and a dependent in dependency parsing), we need to consider *conjunctive features*, i.e., conjunctions of primitive features that focus on the particular clues of each constituent, to achieve a high degree of accuracy in those tasks.

Training with conjunctive features involves a space-time trade-off in the way conjunctive features are handled. Linear models, such as log-linear models, explicitly estimate the weights of conjunctive features, and training thus requires a great deal of memory when we take higher-order conjunctive features into consideration. Kernel-based models such as support vector machines, on the other hand, ensure space efficiency by using the kernel trick to implicitly consider conjunctive features. However, training takes quadratic time in the number of examples, even with online algorithms such as the (kernel) perceptron (Freund and Schapire, 1999), and we cannot fully exploit ample 'labeled' data obtained with semi-supervised algorithms (Ando and Zhang, 2005; Bellare et al., 2007; Liang et al., 2008; Daumé III, 2008).

We aim at resolving this dilemma in training with conjunctive features, and propose online learning that combines the time efficiency of linear training and the space efficiency of kernel-based training. Following the work by Goldberg and Elhadad (2008), we explicitly take conjunctive features into account that frequently appear in the training data, and implicitly consider the other conjunctive features by using the polynomial kernel. We then improve the scalability of this training by a method called *kernel slicing*, which allows us to reuse the temporal margins of partial feature vectors and to terminate computations that do not contribute to parameter updates.

We evaluate our method in two NLP tasks: dependency parsing and hyponymy-relation extraction. We demonstrate that our method is orders of magnitude faster than kernel-based online learning while retaining its space efficiency.

The remainder of this paper is organized as follows. Section 2 introduces preliminaries and notations. Section 3 proposes our training method. Section 4 evaluates the proposed method. Section 5 discusses related studies. Section 6 concludes this paper and addresses future work.

Algorithm 1 BASE LEARNER: KERNEL PA-I

```
INPUT: $\mathcal{T} = \{(\boldsymbol{x}, y)_t\}_{t=1}^{|\mathcal{T}|}, k : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}, C \in \mathbb{R}^+$
OUTPUT: $(\mathcal{S}_{|\mathcal{T}|}, \boldsymbol{\alpha}_{|\mathcal{T}|})$
 1: initialize: $\mathcal{S}_0 \leftarrow \varnothing, \boldsymbol{\alpha}_0 \leftarrow \varnothing$
 2: for $t = 1$ to $|\mathcal{T}|$ do
 3:     receive example $(\boldsymbol{x}, y)_t : \boldsymbol{x} \in \mathbb{R}^n, y \in \{-1, +1\}$
 4:     compute margin: $m_t(\boldsymbol{x}) = \sum_{\boldsymbol{s}_i \in \mathcal{S}_{t-1}} \alpha_i k(\boldsymbol{s}_i, \boldsymbol{x})$
 5:     if $\ell_t = \max\{0, 1 - y m_t(\boldsymbol{x})\} > 0$ then
 6:         $\tau_t \leftarrow \min\left\{C, \frac{\ell_t}{\|\boldsymbol{x}\|^2}\right\}$
 7:         $\boldsymbol{\alpha}_t \leftarrow \boldsymbol{\alpha}_{t-1} \cup \{\tau_t y\}, \mathcal{S}_t \leftarrow \mathcal{S}_{t-1} \cup \{\boldsymbol{x}\}$
 8:     else
 9:         $\boldsymbol{\alpha}_t \leftarrow \boldsymbol{\alpha}_{t-1}, \mathcal{S}_t \leftarrow \mathcal{S}_{t-1}$
10:     end if
11: end for
12: return $(\mathcal{S}_{|\mathcal{T}|}, \boldsymbol{\alpha}_{|\mathcal{T}|})$
```

2 Preliminaries

This section first introduces a passive-aggressive algorithm (Crammer et al., 2006), which we use as a base learner. We then explain fast methods of computing the polynomial kernel.

Each example $\boldsymbol{x}$ in a classification problem is represented by a *feature vector* whose element x_j is a value of a feature function, $f_j \in \mathcal{F}$. Here, we assume a binary feature function, $f_j(\boldsymbol{x}) \in \{0, 1\}$, which returns one if particular context data appear in the example. We say that feature f_j is *active* in example $\boldsymbol{x}$ when $x_j = f_j(\boldsymbol{x}) = 1$. We denote a binary feature vector, $\boldsymbol{x}$, as a set of active features $\boldsymbol{x} = \{f_j \mid f_j \in \mathcal{F}, f_j(\boldsymbol{x}) = 1\}$ for brevity; $f_j \in \boldsymbol{x}$ means that f_j is active in $\boldsymbol{x}$, and $|\boldsymbol{x}|$ represents the number of active features in $\boldsymbol{x}$.

2.1 Kernel Passive-Aggressive Algorithm

A passive-aggressive algorithm (PA) (Crammer et al., 2006) represents online learning that updates parameters for given labeled example $(\boldsymbol{x}, y)_t \in \mathcal{T}$ in each round t. We assume a binary label, $y \in \{-1, +1\}$, here for clarity. Algorithm 1 is a variant of PA (PA-I) that incorporates a kernel function, k. In round t, PA-I first computes a *(signed) margin* $m_t(\boldsymbol{x})$ of $\boldsymbol{x}$ by using the kernel function with support set $\mathcal{S}_{t-1}$ and coefficients $\boldsymbol{\alpha}_{t-1}$ (Line 4). PA-I then suffers a hinge-loss, $\ell_t = \max\{0, 1 - y m_t(\boldsymbol{x})\}$ (Line 5). If $\ell_t > 0$, PA-I adds $\boldsymbol{x}$ to $\mathcal{S}_{t-1}$ (Line 7). Hyperparameter C controls the aggressiveness of parameter updates.

The kernel function computes a dot product in $\mathbb{R}^{\mathcal{H}}$ space without mapping $\boldsymbol{x} \in \mathbb{R}^n$ to $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^{\mathcal{H}}$ ($k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}')$). We can implicitly consider (weighted) d or less order conjunctions of primitive features by using *polynomial kernel function* $k_d(\boldsymbol{s}, \boldsymbol{x}) = (\boldsymbol{s}^{\mathrm{T}}\boldsymbol{x} + 1)^d$. For example, given support vector $\boldsymbol{s} = (s_1, s_2)^{\mathrm{T}}$ and input example $\boldsymbol{x} = (x_1, x_2)^{\mathrm{T}}$, the second-order polynomial kernel returns $k_2(\boldsymbol{s}, \boldsymbol{x}) = (s_1 x_1 + s_2 x_2 + 1)^2 = 1 + 3 s_1 x_1 + 3 s_2 x_2 + 2 s_1 x_1 s_2 x_2$ ($\because s_i, x_i \in \{0, 1\}$). This function thus implies mapping $\boldsymbol{\phi}_2(\boldsymbol{x}) = (1, \sqrt{3} x_1, \sqrt{3} x_2, \sqrt{2} x_1 x_2)^{\mathrm{T}}$.

Although online learning is generally efficient, the kernel spoils its efficiency (Dekel et al., 2008). This is because the kernel evaluation (Line 4) takes $\mathcal{O}(|\mathcal{S}_{t-1}||\boldsymbol{x}|)$ time and $|\mathcal{S}_{t-1}|$ increases as training continues. The learner thus takes the most amount of time in this margin computation.

2.2 Kernel Computation for Classification

This section explains fast, exact methods of computing the polynomial kernel, which are meant to test the trained model, $(\mathcal{S}, \boldsymbol{\alpha})$, and involve substantial computational cost in preparation.

2.2.1 Kernel Inverted

Kudo and Matsumoto (2003) proposed *polynomial kernel inverted* (PKI), which builds inverted indices $h(f_j) \equiv \{\boldsymbol{s} \mid \boldsymbol{s} \in \mathcal{S}, f_j \in \boldsymbol{s}\}$ from each feature f_j to support vector $\boldsymbol{s} \in \mathcal{S}$ to only consider support vector $\boldsymbol{s}$ relevant to given $\boldsymbol{x}$ such that $\boldsymbol{s}^{\mathrm{T}}\boldsymbol{x} \neq 0$. The time complexity of PKI is $\mathcal{O}(B \cdot |\boldsymbol{x}| + |\mathcal{S}|)$ where $B \equiv \frac{1}{|\boldsymbol{x}|} \sum_{f_j \in \boldsymbol{x}} |h(f_j)|$, which is smaller than $\mathcal{O}(|\mathcal{S}||\boldsymbol{x}|)$ if $\boldsymbol{x}$ has many rare features f_j such that $|h(f_j)| \ll |\mathcal{S}|$.

To the best of our knowledge, this is the only exact method that has been used to speed up margin computation in the context of kernel-based online learning (Okanohara and Tsujii, 2007).

2.2.2 Kernel Expansion

Isozaki and Kazawa (2002) and Kudo and Matsumoto (2003) proposed *kernel expansion*, which explicitly maps both support set $\mathcal{S}$ and given example $\boldsymbol{x} \in \mathbb{R}^n$ into $\mathbb{R}^{\mathcal{H}}$ by mapping $\boldsymbol{\phi}_d$ imposed by k_d:

$$m(\boldsymbol{x}) = \left(\sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i \boldsymbol{\phi}_d(\boldsymbol{s}_i) \right)^{\mathrm{T}} \boldsymbol{\phi}_d(\boldsymbol{x}) = \sum_{f_i \in \boldsymbol{x}^d} w_i,$$

where $\boldsymbol{x}^d \in \{0,1\}^{\mathcal{H}}$ is a binary feature vector in which $x_i^d = 1$ for $(\boldsymbol{\phi}_d(\boldsymbol{x}))_i \neq 0$, and $\boldsymbol{w}$ is a weight vector in the expanded feature space, $\mathcal{F}^d$. The weight vector $\boldsymbol{w}$ is computed from $\mathcal{S}$ and $\boldsymbol{\alpha}$:

$$\boldsymbol{w} = \sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i \sum_{k=0}^{d} c_d^k I_k(\boldsymbol{s}_i^d), \qquad (1)$$

where c_d^k is a squared coefficient of k-th order conjunctive features for d-th order polynomial kernel (*e.g.*, $c_2^0 = 1$, $c_2^1 = 3$, and $c_2^2 = 2$)[1] and $I_k(\boldsymbol{s}_i^d)$ is $\boldsymbol{s}_i^d \in \{0,1\}^{\mathcal{H}}$ whose dimensions other than those of k-th order conjunctive features are set to zero.

The time complexity of kernel expansion is $\mathcal{O}(|\boldsymbol{x}^d|)$ where $|\boldsymbol{x}^d| = \sum_{k=0}^{d} \binom{|\boldsymbol{x}|}{k} \propto |\boldsymbol{x}|^d$, which can be smaller than $\mathcal{O}(|\mathcal{S}||\boldsymbol{x}|)$ in usual NLP tasks ($|\boldsymbol{x}| \ll |\mathcal{S}|$ and $d \leq 4$).

2.2.3 Kernel Splitting

Since kernel expansion demands a huge memory volume to store the weight vector, $\boldsymbol{w}$, in $\mathbb{R}^{\mathcal{H}}$ ($\mathcal{H} = \sum_{k=0}^{d} \binom{|\mathcal{F}|}{k}$), Goldberg and Elhadad (2008) only explicitly considered conjunctions among features $f_C \in \mathcal{F}_C$ that commonly appear in support set $\mathcal{S}$, and handled the other conjunctive features relevant to rare features $f_R \in \mathcal{F} \setminus \mathcal{F}_C$ by using the polynomial kernel:

$$\begin{aligned} m(\boldsymbol{x}) &= m(\tilde{\boldsymbol{x}}) + m(\boldsymbol{x}) - m(\tilde{\boldsymbol{x}}) \\ &= \sum_{f_i \in \tilde{\boldsymbol{x}}^d} \tilde{w}_i + \sum_{\boldsymbol{s}_i \in \mathcal{S}_R} \alpha_i k'_d(\boldsymbol{s}_i, \boldsymbol{x}, \tilde{\boldsymbol{x}}), \qquad (2) \end{aligned}$$

where $\tilde{\boldsymbol{x}}$ is $\boldsymbol{x}$ whose dimensions of rare features are set to zero, $\tilde{\boldsymbol{w}}$ is a weight vector computed with Eq. 1 for $\mathcal{F}_C^d$, and $k'_d(\boldsymbol{s}, \boldsymbol{x}, \tilde{\boldsymbol{x}})$ is defined as:

$$k'_d(\boldsymbol{s}, \boldsymbol{x}, \tilde{\boldsymbol{x}}) \equiv k_d(\boldsymbol{s}, \boldsymbol{x}) - k_d(\boldsymbol{s}, \tilde{\boldsymbol{x}}).$$

We can space-efficiently compute the first term of Eq. 2 since $|\tilde{\boldsymbol{w}}| \ll |\boldsymbol{w}|$, while we can quickly compute the second term of Eq. 2 since $k'_d(\boldsymbol{s}_i, \boldsymbol{x}, \tilde{\boldsymbol{x}}) = 0$ when $\boldsymbol{s}_i^{\mathrm{T}} \boldsymbol{x} = \boldsymbol{s}_i^{\mathrm{T}} \tilde{\boldsymbol{x}}$; we only need to consider a small subset of the support set, $\mathcal{S}_R = \bigcup_{f_R \in \boldsymbol{x} \setminus \tilde{\boldsymbol{x}}} h(f_R)$, that has at least one of the rare features, f_R, appearing in $\boldsymbol{x} \setminus \tilde{\boldsymbol{x}}$ ($|\mathcal{S}_R| \ll |\mathcal{S}|$).

Counting the number of features examined, the time complexity of Eq. 2 is $\mathcal{O}(|\tilde{\boldsymbol{x}}^d| + |\mathcal{S}_R||\tilde{\boldsymbol{x}}|)$.

[1]Following Lemma 1 in Kudo and Matsumoto (2003), $c_d^k = \sum_{l=k}^{d} \binom{d}{l} \left(\sum_{m=0}^{k} (-1)^{k-m} \cdot m^l \binom{k}{m} \right)$.

3 Algorithm

This section first describes the way kernel splitting is integrated into PA-I (Section 3.1). We then propose *kernel slicing* (Section 3.2), which enables us to reuse the temporal margins computed in the past rounds (Section 3.2.1) and to skip unnecessary margin computations (Section 3.2.2).

In what follows, we use PA-I as a base learner. Note that an analogous argument can be applied to other perceptron-like online learners with the additive weight update (Line 7 in Algorithm 1).

3.1 Base Learner with Kernel Splitting

A problem in integrating kernel splitting into the base learner presented in Algorithm 1 is how to determine $\mathcal{F}_C$, features among which we explicitly consider conjunctions, without knowing the final support set, $\mathcal{S}_{|\mathcal{T}|}$. We heuristically solve this by ranking feature f according to their frequency in the training data and by using the top-N frequent features in the training data as $\mathcal{F}_C$ ($= \{f \mid f \in \mathcal{F}, \text{RANK}(f) \leq N\}$).[2] Since $\mathcal{S}_{|\mathcal{T}|}$ is a subset of the examples, this approximates the selection from $\mathcal{S}_{|\mathcal{T}|}$. We empirically demonstrate the validity of this approach in the experiments.

We then use $\mathcal{F}_C$ to construct a base learner with kernel splitting; we replace the kernel computation (Line 4 in Algorithm 1) with Eq. 2 where $(\mathcal{S}, \boldsymbol{\alpha}) = (\mathcal{S}_{t-1}, \boldsymbol{\alpha}_{t-1})$. To compute $m_t(\tilde{\boldsymbol{x}})$ by using kernel expansion, we need to additionally maintain the weight vector $\tilde{\boldsymbol{w}}$ for the conjunctions of common features that appear in $\mathcal{S}_{t-1}$.

The additive parameter update of PA-I enables us to keep $\tilde{\boldsymbol{w}}$ to correspond to $(\mathcal{S}_{t-1}, \boldsymbol{\alpha}_{t-1})$. When we add $\boldsymbol{x}$ to support set $\mathcal{S}_{t-1}$ (Line 7 in Algorithm 1), we also update $\tilde{\boldsymbol{w}}$ with Eq. 1:

$$\tilde{\boldsymbol{w}} \leftarrow \tilde{\boldsymbol{w}} + \tau_t y \sum_{k=0}^{d} c_d^k I_k(\tilde{\boldsymbol{x}}^d).$$

Following (Kudo and Matsumoto, 2003), we use a trie (hereafter, *weight trie*) to maintain conjunctive features. Each edge in the weight trie is labeled with a primitive feature, while each path

[2]The overhead of counting features is negligible compared to the total training time. If we want to run the learner in a purely online manner, we can alternatively choose first N features that appear in the processed examples as $\mathcal{F}_C$.

represents a conjunctive feature that combines all the primitive features on the path. The weights of conjunctive features are retrieved by traversing nodes in the trie. We carry out an analogous traversal in updating the parameters of conjunctive features, while registering a new conjunctive feature by adding an edge to the trie.

The base learner with kernel splitting combines the virtues of linear training and kernel-based training. It reduces to linear training when we increase N to $|\mathcal{F}|$, while it reduces to kernel-based training when we decrease N to 0. The output is support set $\mathcal{S}_{|\mathcal{T}|}$ and coefficients $\boldsymbol{\alpha}_{|\mathcal{T}|}$ (optionally, $\tilde{\boldsymbol{w}}$), to which the efficient classification techniques discussed in Section 2.2 and the one proposed by Yoshinaga and Kitsuregawa (2009) can be applied.

Note on weight trie construction The time and space efficiency of this learner strongly depends on the way the weight trie is constructed. We need to address two practical issues that greatly affect efficiency. First, we traverse the trie from the rarest feature that constitutes a conjunctive feature. This rare-to-frequent mining helps us to avoid enumerating higher-order conjunctive features that have not been registered in the trie, when computing margin. Second, we use $\text{RANK}(f)$ encoded into a $\lceil \log_{128} \text{RANK}(f) \rceil$-byte string by using variable-byte coding (Williams and Zobel, 1999) as f's representation in the trie. This encoding reduces the trie size, since features with small $\text{RANK}(f)$ will appear frequently in the trie.

3.2 Base Learner with Kernel Slicing

Although a base learner with kernel splitting can enjoy the merits of linear and kernel-based training, it can simultaneously suffer from their demerits. Because the training takes polynomial time in the number of common features in $\boldsymbol{x}$ ($|\tilde{\boldsymbol{x}}^d| = \sum_{k=0}^{d} \binom{|\tilde{\boldsymbol{x}}|}{k} \propto |\tilde{\boldsymbol{x}}|^d$) at each round, we need to set N to a smaller value when we take higher-order conjunctive features into consideration. However, since the margin computation takes linear time in the number of support vectors $|\mathcal{S}_R|$ relevant to rare features $f_R \in \mathcal{F} \setminus \mathcal{F}_C$, we need to set N to a larger value when we handle a larger number of training examples. The training thereby slows down when we train a classifier with high-order conjunctive features and a large number of training examples.

We then attempt to improve the scalability of the training by exploiting a characteristic of labeled data in NLP. Because examples in NLP tasks are likely to be redundant (Yoshinaga and Kitsuregawa, 2009), the learner computes margins of examples that have many features in common. If we can reuse the 'temporal' margins of partial feature vectors computed in past rounds, this will speed up the computation of margins.

We propose *kernel slicing*, which generalizes kernel splitting in a purely feature-wise manner and enables us to reuse the temporal partial margins. Starting from the most frequent feature f_1 in $\boldsymbol{x}$ ($f_1 = \text{argmin}_{f \in \boldsymbol{x}} \text{RANK}(f)$), we incrementally compute $m_t(\boldsymbol{x})$ by accumulating a *partial margin*, $m_t^j(\boldsymbol{x}) \equiv m_t(\boldsymbol{x}_j) - m_t(\boldsymbol{x}_{j-1})$, when we add the j-th frequent feature f_j in $\boldsymbol{x}$:

$$m_t(\boldsymbol{x}) = m_t^0 + \sum_{j=1}^{|\boldsymbol{x}|} m_t^j(\boldsymbol{x}), \tag{3}$$

where $m_t^0 = \sum_{\boldsymbol{s}_i \in \mathcal{S}_{t-1}} \alpha_i k_d(\boldsymbol{s}_i, \varnothing) = \sum_i \alpha_i$, and $\boldsymbol{x}_j$ has the j most frequent features in $\boldsymbol{x}$ ($\boldsymbol{x}_0 = \varnothing$, $\boldsymbol{x}_j = \bigsqcup_{k=0}^{j-1} \{\text{argmin}_{f \in \boldsymbol{x} \setminus \boldsymbol{x}_k} \text{RANK}(f)\}$).

Partial margin $m_t^j(\boldsymbol{x})$ can be computed by using the polynomial kernel:

$$m_t^j(\boldsymbol{x}) = \sum_{\boldsymbol{s}_i \in \mathcal{S}_{t-1}} \alpha_i k'_d(\boldsymbol{s}_i, \boldsymbol{x}_j, \boldsymbol{x}_{j-1}), \tag{4}$$

or by using kernel expansion:

$$m_t^j(\boldsymbol{x}) = \sum_{f_i \in \boldsymbol{x}_j^d \setminus \boldsymbol{x}_{j-1}^d} \tilde{w}_i. \tag{5}$$

Kernel splitting is a special case of kernel slicing, which uses Eq. 5 for $f_j \in \mathcal{F}_C$ and Eq. 4 for $f_j \in \mathcal{F} \setminus \mathcal{F}_C$.

3.2.1 Reuse of Temporal Partial Margins

We can speed up both Eqs. 4 and 5 by reusing a temporal partial margin, $\delta_{t'}^j = m_{t'}^j(\boldsymbol{x})$ that had been computed in past round $t'(< t)$:

$$m_t^j(\boldsymbol{x}) = \delta_{t'}^j + \sum_{\boldsymbol{s}_i \in \mathcal{S}_j} \alpha_i k'_d(\boldsymbol{s}_i, \boldsymbol{x}_j, \boldsymbol{x}_{j-1}), \tag{6}$$

where $\mathcal{S}_j = \{\boldsymbol{s} \mid \boldsymbol{s} \in \mathcal{S}_{t-1} \setminus \mathcal{S}_{t'-1}, f_j \in \boldsymbol{s}\}$.

Algorithm 2 KERNEL SLICING

```
INPUT: $\boldsymbol{x} \in 2^{\mathcal{F}}, \mathcal{S}_{t-1}, \boldsymbol{\alpha}_{t-1}, \mathcal{F}_C \subseteq \mathcal{F}, \boldsymbol{\delta} : 2^{\mathcal{F}} \mapsto \mathbb{N} \times \mathbb{R}$
OUTPUT: $m_t(\boldsymbol{x})$
 1: initialize: $\boldsymbol{x}_0 \leftarrow \varnothing, j \leftarrow 1, m_t(\boldsymbol{x}) \leftarrow m_t^0$
 2: repeat
 3:     $\boldsymbol{x}_j \leftarrow \boldsymbol{x}_{j-1} \sqcup \{\text{argmin}_{f \in \boldsymbol{x} \setminus \boldsymbol{x}_{j-1}} \text{RANK}(f)\}$
 4:     retrieve partial margin: $(t', \delta_{t'}^j) \leftarrow \boldsymbol{\delta}(\boldsymbol{x}_j)$
 5:     if $f_j \in \mathcal{F} \setminus \mathcal{F}_C$ or Eq. 7 is true then
 6:         compute $m_t^j(\boldsymbol{x})$ using Eq. 6 with $\delta_{t'}^j$
 7:     else
 8:         compute $m_t^j(\boldsymbol{x})$ using Eq. 5
 9:     end if
10:     update partial margin: $\boldsymbol{\delta}(\boldsymbol{x}_j) \leftarrow (t, m_t^j(\boldsymbol{x}))$
11:     $m_t(\boldsymbol{x}) \leftarrow m_t(\boldsymbol{x}) + m_t^j(\boldsymbol{x})$
12: until $\boldsymbol{x}_j \neq \boldsymbol{x}$
13: return $m_t(\boldsymbol{x})$
```

Eq. 6 is faster than Eq. 4,[3] and can even be faster than Eq. 5.[4] When $\text{RANK}(f_j)$ is high, $\boldsymbol{x}_j$ appears frequently in the training examples and $|\mathcal{S}_j|$ becomes small since t' will be close to t. When $\text{RANK}(f_j)$ is low, $\boldsymbol{x}_j$ rarely appears in the training examples but we can still expect $|\mathcal{S}_j|$ to be small since the number of support vectors in $\mathcal{S}_{t-1} \setminus \mathcal{S}_{t'-1}$ that have rare feature f_j will be small.

To compute Eq. 3, we now have the choice to choose Eq. 5 or 6 for $f_j \in \mathcal{F}_C$. Counting the number of features to be examined in computing $m_t^j(\boldsymbol{x})$, we have the following criteria to determine whether we can use Eq. 6 instead of Eq. 5:

$$1 + |\mathcal{S}_j||\boldsymbol{x}_{j-1}| \leq |\boldsymbol{x}_j^d \setminus \boldsymbol{x}_{j-1}^d| = \sum_{k=1}^{d} \binom{j-1}{k-1},$$

where the left- and right-hand sides indicate the number of features examined in Eq. 6 for the former and Eq. 5 for the latter. Expanding the right-hand side for $d = 2, 3$ and dividing both sides with $|\boldsymbol{x}_{j-1}| = j - 1$, we have:

$$|\mathcal{S}_j| \leq \left\{ \begin{array}{ll} 1 & (d = 2) \\ \frac{j}{2} & (d = 3) \end{array} \right. . \qquad (7)$$

If this condition is met after retrieving the temporal partial margin, $\delta_{t'}^j$, we can compute partial margin $m_t^j(\boldsymbol{x})$ with Eq. 6. This analysis reveals that we can expect little speed-up for the second-order polynomial kernel; we will only use Eq. 6 with third or higher-order polynomial kernel.

Algorithm 2 summarizes the margin computation with kernel slicing. It processes each feature $f_j \in \boldsymbol{x}$ in frequent-to-rare order, and accumulates partial margin $m_t^j(\boldsymbol{x})$ to have $m_t(\boldsymbol{x})$. Intuitively speaking, when the algorithm uses the partial margin, it only considers support vectors on each feature that have been added since the last evaluation of the partial feature vector, to avoid the repetition in kernel evaluation as much as possible.

3.2.2 Termination of Margin Computation

Kernel slicing enables another optimization that exploits a characteristic of online learning. Because we need an exact margin, $m_t(\boldsymbol{x})$, only when hinge-loss $\ell_t = 1 - y m_t(\boldsymbol{x})$ is positive, we can finish margin computation as soon as we find that the lower-bound of $y m_t(\boldsymbol{x})$ is larger than one.

When $y m_t(\boldsymbol{x})$ is larger than one after processing feature f_j in Eq. 3, we quickly examine whether this will hold even after we process the remaining features. We can compute a possible range of partial margin $m_t^k(\boldsymbol{x})$ with Eq. 4, having the upper- and lower-bounds, $\hat{k}'_d$ and $\check{k}'_d$, of $k'_d(\boldsymbol{s}_i, \boldsymbol{x}_k, \boldsymbol{x}_{k-1})$ $(= k_d(\boldsymbol{s}_i, \boldsymbol{x}_k) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_{k-1}))$:

$$m_t^k(\boldsymbol{x}) \leq \hat{k}'_d \sum_{\boldsymbol{s}_i \in \mathcal{S}_k^+} \alpha_i + \check{k}'_d \sum_{\boldsymbol{s}_i \in \mathcal{S}_k^-} \alpha_i \qquad (8)$$

$$m_t^k(\boldsymbol{x}) \geq \check{k}'_d \sum_{\boldsymbol{s}_i \in \mathcal{S}_k^+} \alpha_i + \hat{k}'_d \sum_{\boldsymbol{s}_i \in \mathcal{S}_k^-} \alpha_i, \qquad (9)$$

where $\mathcal{S}_k^+ = \{\boldsymbol{s}_i \,|\, \boldsymbol{s}_i \in \mathcal{S}_{t-1}, f_k \in \boldsymbol{s}_i, \alpha_i > 0\}$, $\mathcal{S}_k^- = \{\boldsymbol{s}_i \,|\, \boldsymbol{s}_i \in \mathcal{S}_{t-1}, f_k \in \boldsymbol{s}_i, \alpha_i < 0\}$, $\hat{k}'_d = (k+1)^d - k^d$ and $\check{k}'_d = 2^d - 1$ ($\because 0 \leq \boldsymbol{s}_i^{\mathrm{T}} \boldsymbol{x}_{k-1} \leq |\boldsymbol{x}_{k-1}| = k - 1$, $\boldsymbol{s}_i^{\mathrm{T}} \boldsymbol{x}_k = \boldsymbol{s}_i^{\mathrm{T}} \boldsymbol{x}_{k-1} + 1$ for all $\boldsymbol{s}_i \in \mathcal{S}_k^+ \cup \mathcal{S}_k^-$).

We accumulate Eqs. 8 and 9 from rare to frequent features, and use the intermediate results to estimate the possible range of $m_t(\boldsymbol{x})$ before Line 3 in Algorithm 2. If the lower bound of $y m_t(\boldsymbol{x})$ turns out to be larger than one, we terminate the computation of $m_t(\boldsymbol{x})$.

As training continues, the model becomes discriminative and given $\boldsymbol{x}$ is likely to have a larger margin. The impact of this termination will increase as the amount of training data expands.

[3] When a margin of $\boldsymbol{x}_j$ has not been computed, we regard $t' = 0$ and $\delta_{t'}^j = 0$, which reduces Eq. 6 to Eq. 4.

[4] We associate partial margins with partial feature sequences whose features are sorted by frequent-to-rare order, and store them in a trie (*partial margin trie*). This enables us to retrieve partial margin $\delta_{t'}^j$ for given $\boldsymbol{x}_j$ in $\mathcal{O}(1)$ time.

4 Evaluation

We evaluated the proposed method in two NLP tasks: dependency parsing (Sassano, 2004) and hyponymy-relation extraction (Sumida et al., 2008). We used labeled data included in open-source softwares to promote the reproducibility of our results.[5] All the experiments were conducted on a server with an Intel® Xeon™ 3.2 GHz CPU. We used a double-array trie (Aoe, 1989; Yata et al., 2009) as an implementation of the weight trie and the partial margin trie.

4.1 Task Descriptions

Japanese Dependency Parsing A parser inputs a sentence segmented by a *bunsetsu* (base phrase in Japanese), and selects a particular pair of bunsetsus (dependent and head candidates); the classifier then outputs label $y = +1$ (dependent) or -1 (independent) for the pair. The features consist of the surface form, POS, POS-subcategory and the inflection form of each bunsetsu, and surrounding contexts such as the positional distance, punctuations and brackets. See (Yoshinaga and Kitsuregawa, 2009) for details on the features.

Hyponymy-Relation Extraction A hyponymy relation extractor (Sumida et al., 2008) first extracts a pair of entities from hierarchical listing structures in Wikipedia articles (hypernym and hyponym candidates); a classifier then outputs label $y = +1$ (correct) or -1 (incorrect) for the pair. The features include a surface form, morphemes, POS and the listing type for each entity, and surrounding contexts such as the hierarchical distance between the entities. See (Sumida et al., 2008) for details on the features.

4.2 Settings

Table 1 summarizes the training data for the two tasks. The examples for the Japanese dependency parsing task were generated for a transition-based parser (Sassano, 2004) from a standard data set.[6] We used the dependency accuracy of the parser as model accuracy in this task. In the hyponymy-relation extraction task, we randomly chosen two sets of 10,000 examples from the labeled data for development and testing, and used the remaining examples for training. Note that the number of active features, $|\mathcal{F}^d|$, dramatically grows when we consider higher-order conjunctive features.

DATA SET	DEP	REL
$\lvert\mathcal{T}\rvert$	296,776	201,664
$(y = +1)$	150,064	152,199
$(y = -1)$	146,712	49,465
Ave. of $\lvert\boldsymbol{x}\rvert$	27.6	15.4
Ave. of $\lvert\boldsymbol{x}^2\rvert$	396.1	136.9
Ave. of $\lvert\boldsymbol{x}^3\rvert$	3558.3	798.7
$\lvert\mathcal{F}\rvert$	64,493	306,036
$\lvert\mathcal{F}^2\rvert$	3,093,768	6,688,886
$\lvert\mathcal{F}^3\rvert$	58,361,669	64,249,234

Table 1: Training data for dependency parsing (DEP) and hyponymy-relation extraction (REL).

We compared the proposed method, PA-I SL (Algorithm 1 with Algorithm 2), to PA-I KERNEL (Algorithm 1 with PKI; Okanohara and Tsujii (2007)), PA-I KE (Algorithm 1 with kernel expansion; viz., kernel splitting with $N = |\mathcal{F}|$), SVM (batch training of support vector machines),[7] and ℓ_1-LLM (stochastic gradient descent training of the ℓ_1-regularized log-linear model: Tsuruoka et al. (2009)). We refer to PA-I SL that does not reuse temporal partial margins as PA-I SL*. To demonstrate the impact of conjunctive features on model accuracy, we also trained PA-I without conjunctive features. The number of iterations in PA-I was set to 20, and the parameters of PA-I were averaged in an efficient manner (Daumé III, 2006). We explicitly considered conjunctions among top-N ($N = 125 \times 2^n; n \geq 0$) features in PA-I SL and PA-I SL*. The hyperparameters were tuned to maximize accuracy on the development set.

4.3 Results

Tables 2 and 3 list the experimental results for the two tasks (due to space limitations, Tables 2 and 3 list PA-I SL with parameter N that achieved the fastest speed). The accuracy of the models trained with the proposed method was better than ℓ_1-LLMs and was comparable to SVMs. The infe-

[5]The labeled data for dependency parsing is available from: http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/pecco/, and the labeled data for hyponymy-relation extraction is available from: http://nlpwww.nict.go.jp/hyponymy/.

[6]Kyoto Text Corpus Version 4.0: http://nlp.kuee.kyoto-u.ac.jp/nl-resource/corpus-e.html.

[7]http://chasen.org/~taku/software/TinySVM/

METHOD	d	ACC.	TIME	MEMORY
PA-I	1	88.56%	3s	55MB
ℓ_1-LLM	2	90.55%	340s	1656MB
SVM	2	90.76%	29863s	245MB
PA-I KERNEL	2	90.68%	8361s	84MB
PA-I KE	2	90.67%	41s	155MB
PA-I SL$^*_{N=4000}$	2	90.71%	**33s**	95MB
ℓ_1-LLM	3	90.76%	4057s	21,499MB
SVM	3	90.93%	25912s	243MB
PA-I KERNEL	3	90.90%	8704s	83MB
PA-I KE	3	90.90%	465s	993MB
PA-I SL$_{N=250}$	3	90.89%	**262s**	175MB

Table 2: Training time for classifiers used in dependency parsing task.

METHOD	d	ACC.	TIME	MEMORY
PA-I	1	91.75%	2s	28MB
ℓ_1-LLM	2	92.67%	136s	1683MB
SVM	2	92.85%	12306s	139MB
PA-I KERNEL	2	92.91%	1251s	54MB
PA-I KE	2	92.96%	27s	143MB
PA-I SL$^*_{N=8000}$	2	92.88%	**17s**	77MB
ℓ_1-LLM	3	92.86%	779s	14,089MB
SVM	3	93.09%	17354s	140MB
PA-I KERNEL	3	93.14%	1074s	49MB
PA-I KE	3	93.11%	103s	751MB
PA-I SL$_{N=125}$	3	93.05%	**17s**	131MB

Table 3: Training time for classifiers used in hyponymy-relation extraction task.

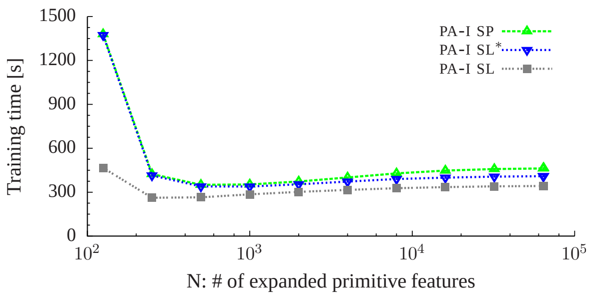

Figure 1: Training time for PA-I variants as a function of the number of expanded primitive features in dependency parsing task ($d = 3$).

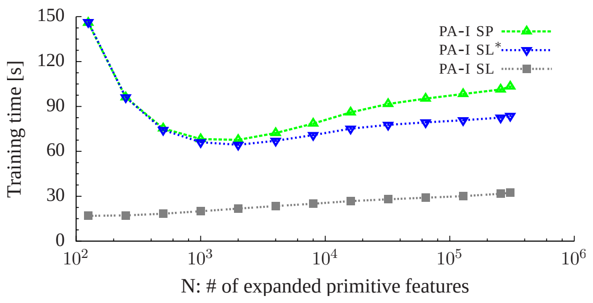

Figure 2: Training time for PA-I variants as a function of the number of expanded primitive features in hyponymy-relation extraction task ($d = 3$).

rior accuracy of PA-I ($d = 1$) confirmed the necessity of conjunctive features in these tasks. The minor difference among the model accuracy of the three PA-I variants was due to rounding errors.

PA-I SL was the fastest of the training methods with the same feature set, and its space efficiency was comparable to the kernel-based learners. PA-I SL could reduce the memory footprint from 993MB[8] to 175MB for $d = 3$ in the dependency parsing task, while speeding up training.

Although linear training (ℓ_1-LLM and PA-I KE) dramatically slowed down when we took higher-order conjunctive features into account, kernel slicing alleviated deterioration in speed. Especially in the hyponymy-relation extraction task, PA-I SL took almost the same time regardless of the order of conjunctive features.

[8]ℓ_1-LLM took much more memory than PA-I KE mainly because ℓ_1-LLM expands conjunctive features in the examples prior to training, while PA-I KE expands conjunctive features in each example on the fly during training. Interested readers may refer to (Chang et al., 2010) for this issue.

Figures 1 and 2 plot the trade-off between the number of expanded primitive features and training time with PA-I variants ($d = 3$) in the two tasks. Here, PA-I SP is PA-I with kernel slicing without the techniques described in Sections 3.2.1 and 3.2.2, viz., kernel splitting. The early termination of margin computation reduces the training time when N is large. The reuse of temporal margins makes the training time stable regardless of parameter N. This suggests a simple, effective strategy for calibrating N; we start the training with $N = |\mathcal{F}|$, and when the learner reaches the allowed memory size, we shrink N to $N/2$ by pruning sub-trees rooted by rarer features with $\text{RANK}(f) > N/2$ in the weight trie.

Figures 3 and 4 plot training time with PA-I variants ($d = 3$) for the two tasks as a function of the training data size. PA-I SP inherited the demerit of PA-I KERNEL which takes quadratic time in the number of examples, while PA-I SL took almost linear time in the number of examples.

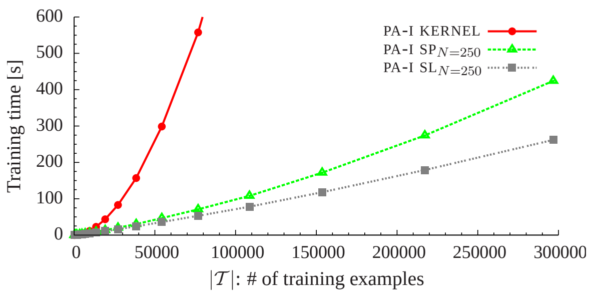

Figure 3: Training time for PA-I variants as a function of the number of training examples in dependency parsing task ($d = 3$).

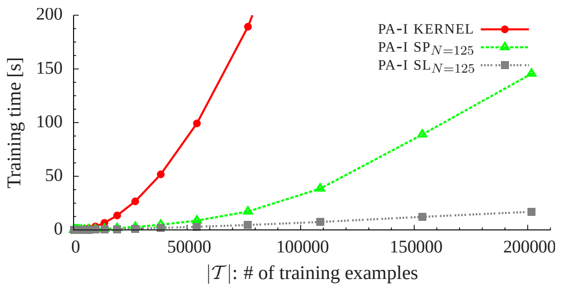

Figure 4: Training time for PA-I variants as a function of the number of training examples in hyponymy-relation extraction task ($d = 3$).

5 Related Work

There are several methods that learn 'simpler' models with fewer variables (features or support vectors), to ensure scalability in training.

Researchers have employed feature selection to assure space-efficiency in linear training. Wu et al. (2007) used frequent-pattern mining to select effective conjunctive features prior to training. Okanohara and Tsujii (2009) revised grafting for ℓ_1-LLM (Perkins et al., 2003) to prune useless conjunctive features during training. Iwakura and Okamoto (2008) proposed a boosting-based method that repeats the learning of rules represented by feature conjunctions. These methods, however, require us to tune the hyperparameter to trade model accuracy and the number of conjunctive features (memory footprint and training time); note that an accurate model may need many conjunctive features (in the hyponymy-relation extraction task, ℓ_1-LLM needed 15,828,122 features to obtain the best accuracy, 92.86%). Our method, on the other hand, takes all conjunctive features into consideration regardless of parameter N.

Dekel et al. (2008) and Cavallanti et al. (2007) improved the scalability of the (kernel) perceptron, by exploiting redundancy in the training data to bound the size of the support set to given threshold B ($\geq |\mathcal{S}_t|$). However, Orabona et al. (2009) reported that the models trained with these methods were just as accurate as a naive method that ceases training when $|\mathcal{S}_t|$ reaches the same threshold, B. They then proposed budget online learning based on PA-I, and it reduced the size of the support set to a tenth with a tolerable loss of accuracy. Their method, however, requires $\mathcal{O}(|\mathcal{S}_{t-1}|^2)$ time in updating the parameters in round t, which disables efficient training. We have proposed an orthogonal approach that exploits the data redundancy in evaluating the kernel to train the same model as the base learner.

6 Conclusion

In this paper, we proposed online learning with kernel slicing, aiming at resolving the space-time trade-off in training a classifier with many conjunctive features. The kernel slicing generalizes kernel splitting (Goldberg and Elhadad, 2008) in a purely feature-wise manner, to truly combine the merits of linear and kernel-based training. To improve the scalability of the training with redundant data in NLP, we reuse the temporal partial margins computed in past rounds and terminate unnecessary margin computations. Experiments on dependency parsing and hyponymy-relation extraction demonstrated that our method could train a classifier orders of magnitude faster than kernel-based learners, while retaining its space efficiency.

We will evaluate our method with ample labeled data obtained by the semi-supervised methods. The implementation of the proposed algorithm for kernel-based online learners is available from http://www.tkl.iis.u-tokyo.ac.jp/˜ynaga/.

Acknowledgment We thank Susumu Yata for providing us practical lessons on the double-array trie, and thank Yoshimasa Tsuruoka for making his ℓ_1-LLM code available to us. We are also indebted to Nobuhiro Kaji and the anonymous reviewers for their valuable comments.

References

Ando, Rie Kubota and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853.

Aoe, Jun'ichi. 1989. An efficient digital search algorithm by using a double-array structure. *IEEE Transactions on Software Engineering*, 15(9):1066–1077.

Bellare, Kedar, Partha Pratim Talukdar, Giridhar Kumaran, Fernando Pereira, Mark Liberman, Andrew McCallum, and Mark Dredze. 2007. Lightly-supervised attribute extraction. In *Proc. NIPS 2007 Workshop on Machine Learning for Web Search*.

Cavallanti, Giovanni, Nicolò Cesa-Bianchi, and Claudio Gentile. 2007. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69(2-3):143–167.

Chang, Yin-Wen, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. 2010. Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11:1471–1490.

Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

Daumé III, Hal. 2006. *Practical Structured Learning Techniques for Natural Language Processing*. Ph.D. thesis, University of Southern California.

Daumé III, Hal. 2008. Cross-task knowledge-constrained self training. In *Proc. EMNLP 2008*, pages 680–688.

Dekel, Ofer, Shai Shalev-Shwartz, and Yoram Singer. 2008. The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing*, 37(5):1342–1372.

Freund, Yoav and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Goldberg, Yoav and Michael Elhadad. 2008. splitSVM: fast, space-efficient, non-heuristic, polynomial kernel computation for NLP applications. In *Proc. ACL-08: HLT, Short Papers*, pages 237–240.

Isozaki, Hideki and Hideto Kazawa. 2002. Efficient support vector classifiers for named entity recognition. In *Proc. COLING 2002*, pages 1–7.

Iwakura, Tomoya and Seishi Okamoto. 2008. A fast boosting-based learner for feature-rich tagging and chunking. In *Proc. CoNLL 2008*, pages 17–24.

Kudo, Taku and Yuji Matsumoto. 2003. Fast methods for kernel-based text analysis. In *Proc. ACL 2003*, pages 24–31.

Liang, Percy, Hal Daumé III, and Dan Klein. 2008. Structure compilation: trading structure for features. In *Proc. ICML 2008*, pages 592–599.

Okanohara, Daisuke and Jun'ichi Tsujii. 2007. A discriminative language model with pseudo-negative samples. In *Proc. ACL 2007*, pages 73–80.

Okanohara, Daisuke and Jun'ichi Tsujii. 2009. Learning combination features with L_1 regularization. In *Proc. NAACL HLT 2009, Short Papers*, pages 97–100.

Orabona, Francesco, Joseph Keshet, and Barbara Caputo. 2009. Bounded kernel-based online learning. *Journal of Machine Learning Research*, 10:2643–2666.

Perkins, Simon, Kevin Lacker, and James Theiler. 2003. Grafting: fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333–1356.

Sassano, Manabu. 2004. Linear-time dependency analysis for Japanese. In *Proc. COLING 2004*, pages 8–14.

Sumida, Asuka, Naoki Yoshinaga, and Kentaro Torisawa. 2008. Boosting precision and recall of hyponymy relation acquisition from hierarchical layouts in Wikipedia. In *Proc. LREC 2008*, pages 2462–2469.

Tsuruoka, Yoshimasa, Jun'ichi Tsujii, and Sophia Ananiadou. 2009. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. In *Proc. ACL-IJCNLP 2009*, pages 477–485.

Williams, Hugh E. and Justin Zobel. 1999. Compressing integers for fast file access. *The Computer Journal*, 42(3):193–201.

Wu, Yu-Chieh, Jie-Chi Yang, and Yue-Shi Lee. 2007. An approximate approach for training polynomial kernel SVMs in linear time. In *Proc. ACL 2007, Interactive Poster and Demonstration Sessions*, pages 65–68.

Yata, Susumu, Masahiro Tamura, Kazuhiro Morita, Masao Fuketa, and Jun'ichi Aoe. 2009. Sequential insertions and performance evaluations for double-arrays. In *Proc. the 71st National Convention of IPSJ*, pages 1263–1264. (In Japanese).

Yoshinaga, Naoki and Masaru Kitsuregawa. 2009. Polynomial to linear: efficient classification with conjunctive features. In *Proc. EMNLP 2009*, pages 1542–1551.