Kernel-based Discriminative Learning Algorithms for Labeling Sequences, Trees, and Graphs

Hisashi Kashima HKASHIMA@JP.IBM.COM
Yuta Tsuboi YUTAT@JP.IBM.COM
IBM Tokyo Research Laboratory, 1623-14 Shimotsuruma, Yamato-shi, 242-8502 Kanagawa, Japan

Abstract

We introduce a new perceptron-based discriminative learning algorithm for labeling structured data such as sequences, trees, and graphs. Since it is fully kernelized and uses pointwise label prediction, large features, including arbitrary number of hidden variables, can be incorporated with polynomial time complexity. This is in contrast to existing labelers that can handle only features of a small number of hidden variables, such as Maximum Entropy Markov Models and Conditional Random Fields. We also introduce several kernel functions for labeling sequences, trees, and graphs and efficient algorithms for them.

1. Introduction

Sequence labeling is one of the important problems widely seen in the areas of natural language processing (NLP), bioinformatics and Web data analysis. Labeling problems generalize supervised classification problems, since not only the label of one hidden variable, but the labels of a set of hidden variables are predicted. Labeling problems for sequences have been extensively studied for years. However, there has been almost no significant work on labeling more general structured data such as trees and graphs. In this paper, we consider kernel-based approaches for labeling problems with general structured data.

Conventionally, in sequence labeling problems, generative models such as Hidden Markov Models (HMMs) have been used. However, they tend to need a lot of data since they target the more difficult problems of estimating joint probabilities of observable variables and hidden variables. Also, they can not handle overlapping features naturally, since observable variables must be independent of each other given the labels of the hidden variables. Recently, conditional models such as Maximum Entropy Markov Models (MEMMs) (McCallum et al., 2000) and Conditional Random Fields (CRFs) (Lafferty et al., 2001) are attracting considerable attention, since they are more suitable for the purpose of predicting the labels of hidden variables given the labels of observable variables. Recently, Collins (2002) proposed the Hidden Markov (HM) Perceptron, a more efficient sequence labeling algorithm based on perceptrons as an alternative to CRFs and MEMMs. SVM-based algorithms (Altun et al., 2003c) and boosting-based algorithms (Altun et al., 2003a) have also been proposed.

In some labeling problems, combinations of local features such as bi-grams are not sufficient for modeling long-distance dependencies such as idioms in NLP, or motifs in bioinformatics. However, in all of the methods proposed so far, there is a fundamental problem that they can handle only local features considering small numbers of hidden variables. Label prediction is based on the Viterbi decoding using dynamic programming, which requires exponential time with respect to the maximum number of hidden variables included in one feature. Although HM-Perceptrons and HM-SVMs have dual form representations, features must be explicitly considered in label prediction. Of special interest here, Kakade et al. (2002) proposed a pointwise log-loss objective function for CRFs and MEMMs that aims to maximize the number of individually correct labels. The important point of their objective function is that it does not need to predict the labels for the entire sequence all at once. In this paper, inspired by the idea of a pointwise log-loss function, we propose *marginalized labeling perceptrons* that solve the label prediction problem by using the marginalized feature vectors. Marginalized labeling perceptrons realize point-wise label prediction, and are fully kernelized by using marginalized kernel functions (Tsuda et al., 2002) with long-distance dependencies among hidden variables.

Also, in this paper, we propose several marginalized kernels used in the kernel marginalized labeling

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

perceptron for labeling various structured data such as sequences, trees, and graphs. Recently, kernel-based methods for classification of structured data have been extensively studied[1], for instance, string kernels (Lodhi et al., 2002; Leslie et al., 2002), tree kernels (Collins & Duffy, 2002; Kashima & Koyanagi, 2002), and graph kernels (Kashima et al., 2003; Gärtner et al., 2003). Most of them are based on feature vectors composed of the counts of the substructures such as subsequences, subtrees, or subgraphs, to incorporate long-distance dependencies. As the dimensionality of feature vectors is typically very high (possibly infinite), they adopt efficient procedures such as suffix trees, dynamic programming, or matrix computation to avoid explicit enumeration of the features. By extending these kernels, we propose several marginalized kernels for labeling sequences, trees, and graphs, and efficient algorithms for computing them. Finally, we show some promising results for experiments on sequence labeling and tree labeling for real-world tasks which require structure information.

Our contributions in this paper are twofold: (i) A fully-kernelized labeling learning algorithm that solves the problem in label prediction for handling features of arbitrary size. (ii) Fast marginalized kernels for labeling sequences, trees and graphs.

2. Hidden Markov Perceptrons

The labeling problem is defined to be the problem of learning a function that maps the observable variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$ to the hidden variables $\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$, where each $x_t \in \Sigma_x$ and each $y_t \in \Sigma_y$. For instance, in part-of-speech tagging, x_t represents the t-th word, and y_t represents the part-of-speech tag of the t-th word (Figure 1). The learner may exploit a set of training examples $E = (e^{(1)}, e^{(2)}, \ldots, e^{(|E|)})$, where $e^{(i)} = (\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ is the i-th example, and $|\boldsymbol{x}^{(i)}| = |\boldsymbol{y}^{(i)}| = T^{(i)}$.

Collins (2002) introduced the HM-Perceptron, a discriminative learning algorithm for sequence labeling as an alternative to CRFs and MEMMs. It can be trained efficiently by processing the training examples one by one. Its ability was shown to be comparable to CRFs (Altun et al., 2003b). The key idea of the HM-Perceptron is to interpret the mapping as a binary classification, i.e. ${\Sigma_x}^T \times {\Sigma_y}^T \to \{+1, -1\}$. Let F be a set of features for vector representation of $(\boldsymbol{x}, \boldsymbol{y})$. Let $\phi_f(\boldsymbol{x}, \boldsymbol{y})$ be the number of times a feature $f \in F$ appears in $(\boldsymbol{x}, \boldsymbol{y})$, and $\Phi(\boldsymbol{x}, \boldsymbol{y})$ be their vector form. Given $\boldsymbol{x}$, the HM-Perceptron outputs the prediction $\hat{\boldsymbol{y}}$ according to the current weights of the features:

$$\begin{aligned} \hat{\boldsymbol{y}} &= \operatorname*{argmax}_{\boldsymbol{y} \in {\Sigma_y}^T} \sum_{f \in F} w_f \phi_f(\boldsymbol{x}, \boldsymbol{y}) \\ &= \operatorname*{argmax}_{\boldsymbol{y} \in {\Sigma_y}^T} \langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}) \rangle, \end{aligned} \quad (1)$$

[1] An extensive survey is found in Gärtner (2003)

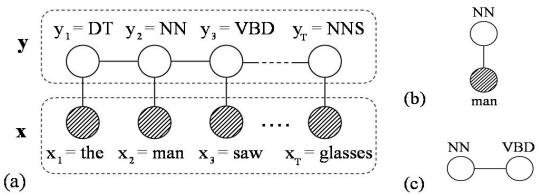

Figure 1. (a) Graphical model representation of a labeled sequence under a first-order Markov assumption. Shadowed nodes indicate observable variables, and white nodes indicate hidden variables. The sentence "the, man, saw ,$\cdots$, glasses." is the label sequence $\boldsymbol{x}$ of the observable variables. The part of speech tag sequence "DT, NN, VBD, ..., NNS" is the label sequence $\boldsymbol{y}$ of the hidden variables. (b) A pair of an observable variable and a hidden variable. (c) A pair of two hidden variables.

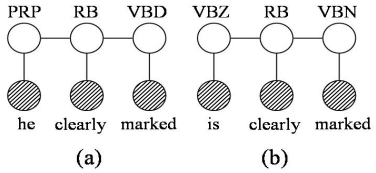

Figure 2. Features for labeling sequences (of length 3). Bigram features are not sufficient for disambiguation of the label of "marked".

where w_f is the weight of a feature f, and $\boldsymbol{w}$ is the vector form of the weights. Under a first-order Markov assumption, two types of features, such as pairs of an observable variable and a hidden variable (Figure 1(b)), and pairs of two hidden variables (Figure 1(c)) are used.

Starting at $\boldsymbol{w} = 0$, the weights are updated by using

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + \Phi(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) - \Phi(\boldsymbol{x}^{(i)}, \hat{\boldsymbol{y}}^{(i)}), \quad (2)$$

when the prediction for $e^{(i)}$ is wrong, i.e. $\hat{\boldsymbol{y}}^{(i)} \neq \boldsymbol{y}^{(i)}$. Equation (1) is also written in a dual form as follows by using the weights of the examples α.

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in {\Sigma_y}^T} \sum_{j=1}^{|E|} \sum_{\tilde{\boldsymbol{y}} \in {\Sigma_y}^T} \alpha_j(\tilde{\boldsymbol{y}}) \big\langle \Phi(\boldsymbol{x}^{(j)}, \tilde{\boldsymbol{y}}), \Phi(\boldsymbol{x}, \boldsymbol{y}) \big\rangle \quad (3)$$

Starting with $\boldsymbol{\alpha} = \boldsymbol{0}$, the updating rules are rewritten as

- $\alpha_i^{new}(\boldsymbol{y}^{(i)}) = \alpha_i^{old}(\boldsymbol{y}^{(i)}) + 1$
- $\alpha_i^{new}(\hat{\boldsymbol{y}}^{(i)}) = \alpha_i^{old}(\hat{\boldsymbol{y}}^{(i)}) - 1,$

when $\hat{\boldsymbol{y}}^{(i)} \neq \boldsymbol{y}^{(i)}$. $(\boldsymbol{x}^{(i)}, \hat{\boldsymbol{y}}^{(i)})$ is called a pseudo-negative example since it acts like an negative example.

Now, we discuss using features that consider long-

distance dependencies in the HM-Perceptron. Sometimes, bi-gram features (Figure 1(b)(c)) are not sufficient for modeling long-distance dependencies such as idioms in NLP, or motifs in bioinformatics. What if we employ longer features such as in Figure 2? Let us assume that we allow features of lengths up to d. Let $\boldsymbol{x}_{t_1}^{t_2}$ be the subset of the observable variables from position t_1 to position t_2 of $\boldsymbol{x}$, and $\boldsymbol{y}_{t_1}^{t_2}$ is defined accordingly. Taking into account that all features depend on at most d consecutive hidden variables, $\Phi(\boldsymbol{x}, \boldsymbol{y})$ is decomposed as

$$\begin{aligned}\Phi(\boldsymbol{x}, \boldsymbol{y}) &= \Phi(\boldsymbol{x}_1^d, \boldsymbol{y}_1^d) - \Phi(\boldsymbol{x}_2^d, \boldsymbol{y}_2^d) \\ &\quad + \Phi(\boldsymbol{x}_2^{d+1}, \boldsymbol{y}_2^{d+1}) - \Phi(\boldsymbol{x}_3^{d+1}, \boldsymbol{y}_3^{d+1}) \\ &\quad \cdots \\ &\quad + \Phi(\boldsymbol{x}_{T-d}^{T-1}, \boldsymbol{y}_{T-d}^{T-1}) - \Phi(\boldsymbol{x}_{T-d+1}^{T-1}, \boldsymbol{y}_{T-d+1}^{T-1}) \\ &\quad + \Phi(\boldsymbol{x}_{T-d+1}^{T}, \boldsymbol{y}_{T-d+1}^{T}).\end{aligned} \tag{4}$$

Substituting Equation (4) into Equation (1), the argmax operation in Equation (1) is performed via the following dynamic programming.

$$\begin{aligned}\max_{\boldsymbol{y}} \big\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}) \big\rangle &= \max_{y_1, \ldots, y_d} s(y_1, \ldots, y_d) \\ s(y_t, \ldots, y_{t+d-1}) &= \max_{y_{t+d}} s(y_{t+1}, \ldots, y_{t+d}) \\ &\quad + \big\langle \boldsymbol{w}, \Phi(\boldsymbol{x}_t^{t+d-1}, \boldsymbol{y}_t^{t+d-1}) - \Phi(\boldsymbol{x}_{t+1}^{t+d-1}, \boldsymbol{y}_{t+1}^{t+d-1}) \big\rangle \\ s(y_{T-d+1}, \ldots, y_T) &= \big\langle \boldsymbol{w}, \Phi(\boldsymbol{x}_{T-d+1}^{T}, \boldsymbol{y}_{T-d+1}^{T}) \big\rangle\end{aligned}$$

However, the computation time in each step of the recursion grows exponentially with respect to the maximum feature length d. Even the dual form (3) still has the same problem, because we have to evaluate all of the features explicitly when predicting labels. This is a serious problem when we want to employ kernels with arbitrarily long features (possibly of infinite length). Also, CRFs have the same problem since they employ dynamic programming procedures based on the same decomposition in learning and prediction. In addition, even the decomposition (4) is not applicable to kernels enumerating the occurrences of features when allowing gaps (Lodhi et al., 2002), since a feature of length of d can occur over an interval of length longer than d. Collins (2000) avoids this problem by a reranking approach where a pre-trained labeler using local features generates an affordable number of candidates for $\boldsymbol{y}$, and the candidates are reranked by using global features. However, the correct answer is not guaranteed to be contained in the candidates based on the local features.

3. Marginalized Labeling Perceptrons

We now propose a new kernel-based labeler that solves the problems of the previous section. Usually, in MEMMs and CRFs, model $P_s(\boldsymbol{y}|\boldsymbol{x})$ is trained by maximizing the sum of the log-likelihood $\sum_i \log P_s(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)})$. This objective function aims to label an entire sequence correctly. On the other hand, Kakade et al. (2002) noticed that it suffices to maximize the number of individually correct labels in many labeling tasks. They proposed to use the marginalized $P_s(\boldsymbol{y}|\boldsymbol{x})$ over all possible assignments of labels for the hidden variables with the t-th hidden variable fixed as $\hat{y}_t \in \Sigma_y$.

$$P_p(y_t = y_t^{(i)}|\boldsymbol{x}^{(i)}) = \sum_{\boldsymbol{y}: y_t = y_t^{(i)}} P_s(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)})$$

The pointwise objective function $\sum_i \sum_t \log P_p(y_t = y_t^{(i)}|\boldsymbol{x}^{(i)})$ is shown to be comparable to the original sequential log-loss function (Altun et al., 2003b). The important point is that label prediction is performed in a pointwise manner since P_p depends only on y_t. We combine this idea with perceptron-based labelers. We propose a *marginalized labeling perceptron* defined as follows.

$$\hat{y}_t = \operatorname*{argmax}_{\tilde{y}_t \in \Sigma_y} \sum_{\boldsymbol{y}: y_t = \tilde{y}_t} P(\boldsymbol{y}|\boldsymbol{x}) \big\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}) \big\rangle. \tag{5}$$

Note that the pointwise label prediction works since the argmax operation depends only on $\tilde{y}_t$. The main idea is that, when y_t is predicted, the original output $\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}) \rangle$ is marginalized over all possible assignments of labels for the hidden variables with the t-th hidden variable fixed as $\hat{y}_t$. $P(\boldsymbol{y}|\boldsymbol{x})$ is some prior distribution over hidden variables that might be pre-trained probabilistic models such as MEMMs or CRFs, or might be designed manually. Since $\sum_{\boldsymbol{y}: y_t = \tilde{y}_t} P(\boldsymbol{y}|\boldsymbol{x}) \Phi(\boldsymbol{x}, \boldsymbol{y})$ can be considered as a new feature vector, the weight vector $\boldsymbol{w}$ is updated by

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + \sum_{\boldsymbol{y}: y_t = y_t^{(i)}} P(\boldsymbol{y}|\boldsymbol{x}) \Phi(\boldsymbol{x}, \boldsymbol{y}) - \sum_{\boldsymbol{y}: y_t = \hat{y}_t^{(i)}} P(\boldsymbol{y}|\boldsymbol{x}) \Phi(\boldsymbol{x}, \boldsymbol{y}) \tag{6}$$

when the prediction for the t-th position of $e^{(i)}$ is wrong, i.e. $\hat{y}_t^{(i)} \neq y_t^{(i)}$.

Next, we derive a dual form of the marginalized labeling perceptron, and obtain a fully-kernelized labeler. Let $\phi_f(\boldsymbol{x}, \boldsymbol{y}; t)$ be the number of times a feature f appears in $(\boldsymbol{x}, \boldsymbol{y})$ with including the t-th position of $(\boldsymbol{x}, \boldsymbol{y})$, and $\Phi_f(\boldsymbol{x}, \boldsymbol{y}; t)$ be the vector form. First, we rewrite the primal form (5) as follows by adding terms that do not depend on $\tilde{y}_t$.

$$\begin{aligned}\hat{y}_t &= \operatorname*{argmax}_{\tilde{y}_t \in \Sigma_y} \sum_{\boldsymbol{y}: y_t = \tilde{y}_t} P(\boldsymbol{y}|\boldsymbol{x}) \big\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}) \big\rangle \\ &\quad + \sum_{\tilde{y}_t \in \Sigma_y} \sum_{\boldsymbol{y}: y_t = \tilde{y}_t} P(\boldsymbol{y}|\boldsymbol{x}) \big\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}; t) \big\rangle \\ &\quad - \sum_{\boldsymbol{y} \in {\Sigma_y}^T} P(\boldsymbol{y}|\boldsymbol{x}) \big\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}) \big\rangle \\ &= \operatorname*{argmax}_{\tilde{y}_t \in \Sigma_y} \sum_{\boldsymbol{y}: y_t = \tilde{y}_t} P(\boldsymbol{y}|\boldsymbol{x}) \big\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}; t) \big\rangle.\end{aligned} \tag{7}$$

```
// Initialization
α := 0
// Training
for round = 1, ..., max_round,
  for i = 1, ..., |E|,
    // kernel computation
    for j = 1, ..., |E|,
      for t = 1, ..., T^(i), and τ = 1, ..., T^(j),
        for y̆_τ ∈ Σ_y, and ỹ_τ ∈ Σ_y,
          compute K(x^(j), x^(i), τ, t, y̆_τ, ỹ_t)
        end_for
      end_for
    end_for
    // prediction & update
    for t = 1, ..., T^(i),
      predict ŷ_t^(i) by using Equation (9)
      if ŷ_t^(i) ≠ y_t^(i),
        α_it(y_t^(i)) := α_it(y_t^(i)) + 1
        α_it(ŷ_t^(i)) := α_it(ŷ_t^(i)) − 1
      end_if
    end_for
  end_for
end_for
```

Figure 3. Kernel Marginalized Labeling Perceptron

The update rule is then rewritten as

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + \sum_{\boldsymbol{y}:y_t=y_t^{(i)}} P(\boldsymbol{y}|\boldsymbol{x})\Phi(\boldsymbol{x},\boldsymbol{y};t) - \sum_{\boldsymbol{y}:y_t=\hat{y}_t^{(i)}} P(\boldsymbol{y}|\boldsymbol{x})\Phi(\boldsymbol{x},\boldsymbol{y};t).$$

By introducing example weights $\boldsymbol{\alpha}$, $\boldsymbol{w}$ is rewritten as a linear combination,

$$\boldsymbol{w} = \sum_{j=1}^{|E|}\sum_{\tau=1}^{T^{(j)}}\sum_{\breve{y}_\tau\in\Sigma_y}\alpha_{j\tau}(\breve{y}_\tau)\sum_{\boldsymbol{y}:y_\tau=\breve{y}_\tau} P(\boldsymbol{y}|\boldsymbol{x})\Phi(\boldsymbol{x}^{(j)},\boldsymbol{y};\tau). \quad (8)$$

Finally, substituting Equation (8) into Equation (7), we obtain the *kernel marginalized labeling perceptron*,

$$\hat{y}_t = \operatorname*{argmax}_{\tilde{y}_t\in\Sigma_y}\sum_{j=1}^{|E|}\sum_{\tau=1}^{T^{(j)}}\sum_{\breve{y}_\tau\in\Sigma_y}\alpha_{j\tau}(\breve{y}_\tau)K(\boldsymbol{x}^{(j)},\boldsymbol{x},\tau,t,\breve{y}_\tau,\tilde{y}_t), \quad (9)$$

where

$$K(\boldsymbol{x}^{(j)},\boldsymbol{x},\tau,t,\breve{y}_\tau,\tilde{y}_t) = \quad (10)$$
$$\sum_{\boldsymbol{y}:y_\tau=\breve{y}_\tau}\sum_{\boldsymbol{y}':y'_t=\tilde{y}_t} P(\boldsymbol{y}|\boldsymbol{x}^{(j)})P(\boldsymbol{y}'|\boldsymbol{x})\big\langle\Phi(\boldsymbol{x}^{(j)},\boldsymbol{y};\tau),\Phi(\boldsymbol{x},\boldsymbol{y}';t)\big\rangle$$

is called the marginalized kernel since the kernel is marginalized over all possible assignments of labels for hidden variables with fixed labels at τ and t. The algorithm is described in Figure 3.

4. Kernel Computation

In this section, we propose several instances of the marginalized kernel (10) for labeling sequences, ordered trees, and directed acyclic graphs. Recently, kernels for structured data such as sequences, trees, and graphs have been studied actively (Gärtner, 2003). We extend these kernels for labeling problems. In all kernels, for the sake of simplicity, we suppose that the prior $P(\boldsymbol{y}|\boldsymbol{x})$ can be decomposed as

$$P(\boldsymbol{y}|\boldsymbol{x}) = \prod_{t=1}^{T} P(y_t|x_t).$$

However, the following discussion can be generalized to exploit more general models like CRFs. In the remainder of this section, we use $K(\tau,t,\breve{y}_\tau,\tilde{y}_t)$ to refer the kernel function given i and j. The important point in computing our kernels is that all features can be constructed by combining smaller features. For instance, in Figure 4, suppose that the feature (b) appears with its rightmost position at t, and the feature (c) appears with its leftmost position at t. Then, the feature (a), the combination of the two features, appears with its second variable at t. All the marginalized kernels that we propose in this paper are decomposed into the upstream kernels $K_U(\tau,t)$, the downstream kernels $K_D(\tau,t)$, and the pointwise kernels $K_P(\tau,t,\breve{y}_\tau,\tilde{y}_t)$ as follows.

$$K(\tau,t,\breve{y}_\tau,\tilde{y}_t) = K_U(\tau,t)\cdot K_P(\tau,t,\breve{y}_\tau,\tilde{y}_t)\cdot K_D(\tau,t)$$

$$K_U(\tau,t) = \sum_{\boldsymbol{y}_U(\tau)}\sum_{\boldsymbol{y}'_U(t)} P(\boldsymbol{y}_U(\tau)|\boldsymbol{x}_U^{(j)}(\tau))P(\boldsymbol{y}'_U(t)|\boldsymbol{x}_U^{(i)}(t))$$
$$\cdot\big\langle\Phi(\boldsymbol{x}_U^{(j)}(\tau),\boldsymbol{y}_U(\tau);\tau),\Phi(\boldsymbol{x}_U^{(i)}(t),\boldsymbol{y}'_U(t);t)\big\rangle \quad (11)$$

$$K_D(\tau,t) = \sum_{\boldsymbol{y}_D(\tau)}\sum_{\boldsymbol{y}'_D(t)} P(\boldsymbol{y}_D(\tau)|\boldsymbol{x}_D^{(j)}(\tau))P(\boldsymbol{y}'_D(t)|\boldsymbol{x}_D^{(i)}(t))$$
$$\cdot\big\langle\Phi(\boldsymbol{x}_D^{(j)}(\tau),\boldsymbol{y}_D(\tau);\tau),\Phi(\boldsymbol{x}_D^{(i)}(t),\boldsymbol{y}'_D(t);t)\big\rangle \quad (12)$$

$$K_P(\tau,t,\breve{y}_\tau,\tilde{y}_t) = \frac{P(\breve{y}_\tau|x_\tau^{(j)})P(\tilde{y}_t|x_t^{(i)})\big\langle\Phi(x_\tau^{(j)},\breve{y}_\tau;\tau),\Phi(x_t^{(i)},\tilde{y}_t;t)\big\rangle}{\Big(\sum_{\breve{y}_\tau}\sum_{\tilde{y}_t} P(\breve{y}_\tau|x_\tau^{(j)})P(\tilde{y}_t|x_t^{(i)})\big\langle\Phi(x_\tau^{(j)},\breve{y}_\tau;\tau),\Phi(x_t^{(i)},\tilde{y}_t;t)\big\rangle\Big)^2} \quad (13)$$

where $\boldsymbol{y}_U(\tau)$ is the set of the hidden variables lying upstream of τ, $\boldsymbol{y}_D(\tau)$ is the set of the hidden variables lying downstream of τ, and $\boldsymbol{y}_U(\tau)\cap\boldsymbol{y}_D(\tau) = \{y_\tau\}$. The denominator of $K_P(\tau,t,\breve{y}_\tau,\tilde{y}_t)$ cancels the overlap of $K_U(\tau,t)$ and $K_D(\tau,t)$. Although explicit computation of the marginalization in K_U and K_D is prohibitive, we can efficiently compute them by using dynamic programming in many cases. The major advantage of the decomposition is that, given the i-th example and the j-th examples, marginalized kernels for all possible values of $(\tau,t,\breve{y}_\tau,\tilde{y}_t)$ can be computed at the same time.

4.1. Sequence Labeling

Now we introduce kernels for sequence labeling. To consider the long-distance dependencies in sequences,

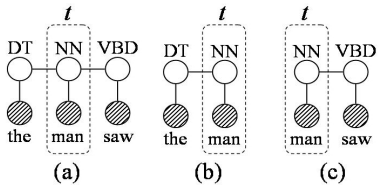

Figure 4. Combining upstream features and downstream features to make larger features. Appearance of the feature (a) with its second variable at t is guaranteed by the appearances of the feature (b) with its rightmost position at t and the feature (c) with its leftmost position at t.

we define each feature as a consecutive pair of a hidden variable and an observable variable of arbitrary length, as in Figure 2. Since the lengths of the features are not restricted, and features of various lengths are mixed, we vary the weight ϕ_f according to the length of the feature f by using the parameter $c > 0$. If the length of the feature is d, and if we observe the feature N times in $(\boldsymbol{x}, \boldsymbol{y})$, $\phi_f(\boldsymbol{x}, \boldsymbol{y})$ is defined as $N \cdot c^d$.
$K_U(\tau, t)$ for labeling sequences is the kernel that considers only the appearances of features with their rightmost positions at τ and t, and which is marginalized over all possible assignments for $\boldsymbol{y}_U(\tau)$ and $\boldsymbol{y}_U(t)$. Similarly, $K_D(\tau, t)$ is the kernel that considers only the appearances of features with their leftmost positions at τ and t, and which is marginalized over all possible assignments for $\boldsymbol{y}_D(\tau)$ and $\boldsymbol{y}_D(t)$.
The algorithm consists of two dynamic programming loops, one for K_U, and the other for K_D, just like the forward-backward algorithm.

$$K_U(\tau, t) = c^2 k(x_\tau^{(j)}, x_t^{(i)})(K_U(\tau - 1, t - 1) + 1) \quad (14)$$

$$K_D(\tau, t) = c^2 k(x_\tau^{(j)}, x_t^{(i)})(K_D(\tau + 1, t + 1) + 1), \quad (15)$$

where

$$k(x_\tau^{(j)}, x_t^{(i)}) = \sum_{\breve{y}_\tau \in \Sigma_y} \sum_{\tilde{y}_t \in \Sigma_y} P(\breve{y}_\tau | x_\tau^{(j)}) P(\tilde{y}_t | x_t^{(i)}) \cdot k_y(\breve{y}_\tau, \tilde{y}_t) k_x(x_\tau^{(j)}, x_t^{(i)})$$

Note that $K_U(\tau, 0) = K_U(0, t) = K_D(\tau, |T^{(i)}| + 1) = K_D(|T^{(j)}| + 1, t) = 0$. K_P is computed as follows.

$$K_P(\tau, t, \breve{y}_\tau, \tilde{y}_t) = \frac{c^2 P(\breve{y}_\tau | x_\tau^{(j)}) P(\tilde{y}_t | x_t^{(i)}) k_y(\breve{y}_\tau, \tilde{y}_t) k_x(x_\tau^{(j)}, x_t^{(i)})}{\left(c^2 k(x_\tau^{(j)}, x_t^{(i)})\right)^2}, \quad (16)$$

where k_x and k_y are functions that return 1 when the arguments are identical, and return 0 elsewhere. However, we can replace them with kernels between the two variables. Apparently, the time complexity of computing the kernels is $O(T^{(i)} T^{(j)})$.[2]

[2]Gappy matching as in Lodhi et al. (2002) is allowed by introducing a penalty parameter $0 \leq \lambda \leq 1$. An appearance of a feature is counted as λ^g if g gaps are inserted. The recursive equation (14) is modified by using the accumulated upstream kernel S_F as follows.

$$\begin{aligned} K_U(\tau, t) &= c^2 k(x_\tau^{(j)}, x_t^{(i)})(S_U(\tau - 1, t - 1) + 1) \\ S_U(\tau, t) &= K_U(\tau, t) + \lambda S_U(\tau, t - 1) \\ &\quad + \lambda S_U(\tau - 1, t) + \lambda^2 S_U(\tau - 1, t - 1) \end{aligned}$$

The downstream kernel (15) is modified accordingly.

4.2. Tree Labeling

The kernel for labeling ordered trees (Figure 5) is based on the labeled ordered tree kernel (Kashima & Koyanagi, 2002). We allow arbitrary tree-structured features such as in Figure 5. By using the mixing parameter c as in the sequence labeling, $\phi_f(\boldsymbol{x}, \boldsymbol{y})$ is defined to be $N \cdot c^d$ where N is the number of times the feature f appears as subgraphs in $(\boldsymbol{x}, \boldsymbol{y})$, and d is the size of f. $K_U(\tau, t)$ for labeling trees is the kernel that considers only the appearances of features with their leaf positions at τ and t, and that is marginalized over all possible assignments for $\boldsymbol{y}_U(\tau)$ and $\boldsymbol{y}_U(t)$. Similarly, $K_D(\tau, t)$ is the kernel that considers only appearances of features with their root positions at τ and t, and that is marginalized over all possible assignments for $\boldsymbol{y}_D(\tau)$ and $\boldsymbol{y}_D(t)$.
The algorithm consists of two dynamic programming loops, one for K_U, and the other for K_D, just as in the inside-outside algorithm. Let $ch(\tau, \upsilon)$ be the index of the υ-th child node of the τ-th node, $pa(\tau)$ be the index of the parent node of the τ-th node, $\#ch(\tau)$ be the number of the child nodes of the τ-th node, and $chID(\tau)$ be the index that means the τ-th node is the $chID(\tau)$-th child of its parent node.
Similar to the recursive computation of the labeled ordered tree kernel (Kashima & Koyanagi, 2002), K_D is computed by dynamic programming in a post-order traversal by using

$$K_D(\tau, t) = c^2 k(x_\tau^{(j)}, x_t^{(i)}) S_F(\tau, t, \#ch(\tau), \#ch(t)),$$

where S_F is also defined recursively as follows,

$$\begin{aligned} S_F(\tau, t, \upsilon, u) = {} & K_D(ch(\tau, \upsilon), ch(t, u)) S_F(\tau, t, \upsilon - 1, u - 1) \\ & + S_F(\tau, t, \upsilon - 1, u) + S_F(\tau, t, \upsilon, u - 1) \\ & - S_F(\tau, t, \upsilon - 1, u - 1), \end{aligned}$$

where $S_F(\tau, t, 0, u) = S_F(\tau, t, \upsilon, 0) = 1$. Intuitively, $S_F(\tau, t, \upsilon, u)$ is the sum of the contributions of all the way of matching between the nodes indexed from $ch(\tau, 1)$ to $ch(\tau, \upsilon)$ and the nodes indexed from $ch(t, 1)$ to $ch(t, u)$.
At the same time, for the sake of the computation of K_U, we also define $S_B(\tau, t, \upsilon, u)$, the sum of the contributions of all the way of matching between the nodes indexed from $ch(\tau, \upsilon)$ to $ch(\tau, \#ch(\tau))$ and the nodes indexed from $ch(t, u)$ to $ch(t, \#ch(t))$, as

$$\begin{aligned} S_B(\tau, t, \upsilon, u) = {} & K_D(ch(\tau, \upsilon), ch(t, u)) S_B(\tau, t, \upsilon + 1, u + 1) \\ & + S_B(\tau, t, \upsilon + 1, u) + S_B(\tau, t, \upsilon, u + 1) \\ & - S_B(\tau, t, \upsilon + 1, u + 1), \end{aligned}$$

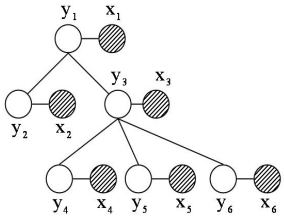

Figure 5. Ordered trees: shadowed nodes indicate observable variables, and white nodes indicate hidden variables.

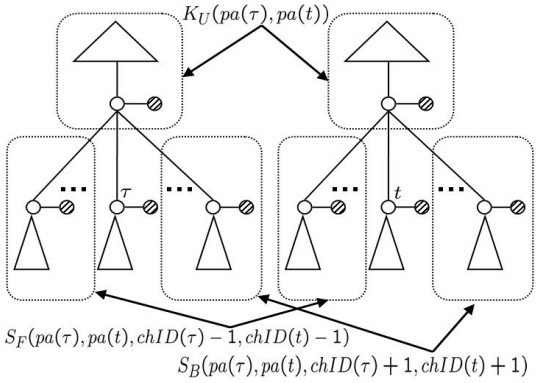

Figure 6. Intuitive image for computing $K_U(\tau, t)$.

where $S_B(\tau, t, \#ch(\tau)+1, u) = S_B(\tau, t, v, \#ch(t)+1) = 1$.
K_U is computed by dynamic programming in a preorder traversal. $K_U(\tau, t)$ is computed by combining the upstream kernel of the parents and the downstream kernels of the siblings (Figure 6),

$$\begin{aligned} K_U(\tau,t) &= c^2 k(x_\tau^{(j)}, x_t^{(i)}) \Big(1 + K_U(pa(\tau), pa(t)) \\ &\quad \cdot S_F(pa(\tau), pa(t), chID(\tau)-1, chID(t)-1) \\ &\quad \cdot S_B(pa(\tau), pa(t), chID(\tau)+1, chID(t)+1)\Big), \end{aligned}$$

Note that the pointwise kernel K_P is the same as in Equation (16). A similar analysis to that of Kashima and Koyanagi (2002) can show that the time complexity of computing the kernel is $O(T^{(i)}T^{(j)})$.

4.3. Graph Labeling

Finally, we propose a kernel for labeling directed acyclic graphs (DAGs) (Figure 7). Unfortunately, it has been shown to be NP-hard to use arbitrary graph-structured features (Gärtner et al., 2003). Therefore, we propose a marginalized kernel using directed path features like Figure 2, based on the DAG kernel (Schölkopf et al., 2004).
$K_U(\tau, t)$ for labeling DAGs is the kernel that considers only the appearances of features with their end positions at τ and t, and which is marginalized over all

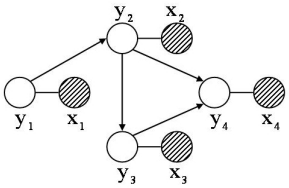

Figure 7. Directed acyclic graphs: shadowed nodes indicate observable variables, and white nodes indicate hidden variables.

possible assignments for $\boldsymbol{y}_U(\tau)$ and $\boldsymbol{y}_U(t)$. Similarly, $K_D(\tau, t)$ is the kernel that considers only the appearances of features with their start positions at τ and t, and which is marginalized over all possible assignments for $\boldsymbol{y}_D(\tau)$ and $\boldsymbol{y}_D(t)$. As in the kernels for sequence labeling, we can write K_U and K_D recursively,

$$K_U(\tau,t) = c^2 k(x_\tau^{(j)}, x_t^{(i)}) \left(1 + \sum_{v \in Pa(\tau)} \sum_{u \in Pa(t)} K_U(v,u)\right)$$

$$K_D(\tau,t) = c^2 k(x_\tau^{(j)}, x_t^{(i)}) \left(1 + \sum_{v \in Ch(\tau)} \sum_{u \in Ch(t)} K_D(v,u)\right),$$

where $Pa(\tau)$ and $Ch(\tau)$ are the set of indices of the parent nodes and the child nodes of the τ-th node, respectively. Note that the pointwise kernel K_P is the same as Equation (16). The time complexity of computing this kernel is $O\big(|T^{(i)}||T^{(j)}|(\max_{\tau,t}|Pa(\tau)||Pa(t)| + \max_{\tau,t}|Ch(\tau)||Ch(t)|)\big)$.

5. Experiments

Finally, we will show the result for experiments on NLP data to assess the importance of large structural features in a real-life situation. We conducted two experiments on Named Entity Recognition (NER) and Product Usage Information Extraction tasks. We compared the performance of the kernel marginalized labeling perceptron with the HM-Perceptrons, which is limited to smaller features in nature.

5.1. Named Entity Recognition

NER is a kind of information extraction task that deals with identifying proper names such as person names and organization names in sentences. We used a subcorpus consisting of the first 300 sentences (8,541 tokens) from the Spanish corpus provided for the Special Session of CoNLL2002 on NER. The task is to label each word with one of the nine labels, i.e. $|\Sigma_y| = 9$, that represents the types and boundaries of named entities, including a dummy label for non-named entities. The kernel between two observable labels is designed by using words and their spelling features, which are described in Altun et al. (2003b) as the S2 features.

Table 1. Named Entity Recognition task result

	Accuracy	Precision	Recall	F1
Sequence Kernel	**88.4**% (3.9)	**52.3**% (17.5)	**19.3**% (2.2)	**27.9** (5.1)
HM-Perceptron	82.9% (7.4)	21.8% (9)	15.6%(4.5)	17.2 (4)

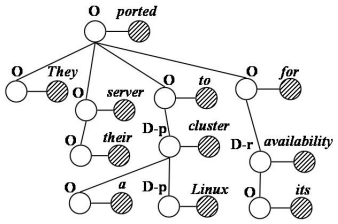

Figure 8. An example of lexicalized dependency trees for the sentence "They ported their server to a linux cluster for its availability" where the hidden variable label "D-p" is for the name of products which a customer uses, "D-r" is for the reason of use, and "O" is dummy label.

In both algorithms, it is combined with a second degree polynomial kernel. A window of size 3 is also used in HM-Perceptrons according to (Altun et al., 2003c). For the proposed perceptron algorithm, we used the sequence kernel withthe mixing parameter $c = 1$ which, was determined in preliminary experiments. The prior over hidden variables is modeled as a uniformed distribution, i.e. $P(y_t|x_t) = 1/|\Sigma_y|$. The performances are evaluated according to the labeling accuracies (including correct answers for dummy labels), precision and recall of named entity labels, and the F1 measure which is the harmonic mean of the precision and recall. Table 1 shows the results of the NER task in a 3-fold cross-validation. The values in parentheses represent the standard deviations on each cross-validation. The proposed perceptron algorithm with the sequence kernel outperforms the other method. Note that the F1 measure score is a more suitable measure for evaluating NER tasks than the labeling accuracy because the majority of the annotated labels are dummy labels. The good performance of the proposed sequence kernel is due to the nature of NER tasks. NER tasks are known as one of the NLP problems which require considering a wide context. For instance, the same phrase "White house" can be a location or an organization entity in different contexts. This empirical result supports the advantages of the fully-kernelized labeler that can handle rich contextual information efficiently without any manual selection of an appropriate feature size.

5.2. Product Usage Information Extraction

Product Usage Information Extraction is a special case of NER tasks which deals with extracting information about the usage of products from a sales log written by sales representatives. We used 184 sentences (3,570 tokens) from an annotated sales log written in Japanese. This task aims to identify the product name (p), its vendor (v), the number of products bought (n), and the reason for buying (r) linked to a classification of the customer state that indicates whether the costomer is already using (D), wishes to use (P), or rejects (R) using the products. In total, there are $|\Sigma_y| = 12$ different labels including a dummy label (O).

All of the Japanese sentences were previously segmented into words, and these words were annotated with the part-of-speech and base form. In all of the algorithms, we designed the kernel between two observable labels by using words, part-of-speech tags, base forms, and character type features. We used it by combining with the second degree polynomial kernel.

In this task, we use not only the sequence kernel for the proposed perceptron, but also the tree kernel. For the tree kernel, we used (lexicalized) dependency trees that represent linguistic structures of the sentences in terms of the dependencies between words. Figure 8 shows an example of a dependency tree in English. To build the dependency trees for the data set, we used a Japanese statistical parser (Kanayama et al., 2000).[3] For both kernels, the mixing parameter is $c = 0.8$.

Table 2. Product Usage Information Extraction task result

	Accuracy	Precision	Recall	F1
Sequence Kernel	88.4% (1.4)	45.7% (10)	**35.2**% (17.7)	36.7 (5.8)
Tree Kernel	**89.8**% (1.2)	**51.5**% (5.2)	32.3% (17.4)	**37.9** (12.1)
HM-Perceptron	87.7%(2.0)	40%(16.5)	29%(11.4)	30.7(9.3)

Table 2 shows the result of the Product Usage Information Extraction task in 3-fold cross-validation. The parameter settings and the evaluation measures are the same as in the NER task. Again, we can see the advantage of the proposed perceptron algorithm against the HM-Perceptrons. Furthermore, the tree kernel utilizing dependency trees shows higher performance than the sequence kernel, which elaborates the advantage of using structured information.

6. Conclusion

In this paper, we introduced an efficient fully-kernelized learning algorithm for labeling structured data such as sequences, trees, and graphs. Our approach can handle large features including arbitrary numbers of variables by using the pointwise label prediction inspired by Kakade et al. (2002), and the marginalized kernels. We also proposed several instances of the marginalized kernels for labeling sequences, ordered trees and directed acyclic graphs. In the preliminary experiments on information extraction

[3]The accuracy of the parser is 88%.

tasks using real-world data, our approach was shown to be promising.

As a related work, Weston et al. (2003) have proposed another general framework that aims at learning the direct mapping from $\Phi_x(\boldsymbol{x})$ to $\Phi_y(\boldsymbol{y})$ with arbitrary kernels. However, features depending on both $\boldsymbol{x}$ and $\boldsymbol{y}$ are not used. Also, they provide no efficient way to decode $\Phi_y(\boldsymbol{y})$ into $\boldsymbol{y}$.

One possible extension of this research is to incorporate more complicated prior distributions such as MEMMs and CRFs in the marginalized kernels, although we employed a very simple form of prior distribution in this paper. It is an interesting question whether our approach boosts such probabilistic models.

Another extension might be an SVM version of our approach. Since HM-SVMs (Altun et al., 2003c) consider an entire label sequence as an example, maximization of the margin between correct label sequences and the second best label sequences is not always acceptable in problems where each sequence has some difficult labels. On the other hand, our approach avoids this problem by considering a label at each position as an individual example.

References

Altun, Y., Hofmann, T., & Johnson, M. (2003a). Discriminative learning for label sequences via boosting. *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press.

Altun, Y., Johnson, M., & Hofmann, T. (2003b). Investigating loss functions and optimization methods for discriminative learning of label sequences. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Altun, Y., Tsochantaridis, I., & Hofmann, T. (2003c). Hidden Markov support vector machines. *Proceedings of the Twentieth International Conference on Machine Learning*.

Collins, M. (2000). Discriminative reranking for natural language parsing. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 175–182). San Francisco, CA: Morgan Kaufmann.

Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Collins, M., & Duffy, N. (2002). Convolution kernels for natural language. *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.

Gärtner, T. (2003). A survey of kernels for structured data. *SIGKDD Explorations*, *5*, S268–S275.

Gärtner, T., Flach, P., & Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory*.

Kakade, S., Teh, Y. W., & Roweis, S. (2002). An alternative objective function for Markovian fields. *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 275–282). San Francisco, CA: Morgan Kaufmann.

Kanayama, H., Torisawa, K., Mitsuishi, Y., & Tsujii, J. (2000). A hybrid Japanese parser with handcrafted grammar and statistics. *Proceedings of the 18th International Conference on Computational Linguistics* (pp. 411–417).

Kashima, H., & Koyanagi, T. (2002). Kernels for semi-structured date. *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 291–298). San Francisco, CA: Morgan Kaufmann.

Kashima, H., Tsuda, K., & Inokuchi, A. (2003). Marginalized kernels between labeled graphs. *Proceedings of the Twentieth International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the Eighteenth International Conference on Machine Learning* (pp. 282–289). San Francisco, CA: Morgan Kaufmann.

Leslie, C., Eskin, E., & Noble, W. S. (2002). The spectrum kernel: A string kernel for SVM protein classification. *Proceedings of the Pacific Symposium on Biocomputing* (pp. 566–575). World Scientific.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, *2*, 419–444.

McCallum, A., D.Freitag, & Pereira, F. (2000). Maximum entropy Markov models for information extraction and segmentation. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 591–598). San Francisco, CA: Morgan Kaufmann.

Schölkopf, B., Tsuda, K., & Vert, J.-P. (Eds.). (2004). *Kernel Methods in Bioinformatics*. Cambridge, MA: MIT Press, to appear.

Tsuda, K., Kin, T., & Asai, K. (2002). Marginalized kernels for biological sequences. *Bioinformatics*, *18*, S268–S275.

Weston, J., Chapelle, O., Elisseeff, A., Schölkopf, B., & Vapnik, V. (2003). Kernel dependency estimation. *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press.