Structured Output Learning with Polynomial Kernel

Hajime Morita
Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology
morita@lr.pi.titech.ac.jp

Hiroya Takamura
Precision and Intelligence Laboratory, Tokyo Institute of Technology
takamura@pi.titech.ac.jp

Manabu Okumura
Precision and Intelligence Laboratory, Tokyo Institute of Technology
oku@pi.titech.ac.jp

Abstract

We propose a new method which enables the training of a kernelized structured output model. The structured output learning can flexibly represent a problem, and thus is gaining popularity in natural language processing. Meanwhile the polynomial kernel method is effective in many natural language processing tasks, since it takes into account the combination of features. However, it is computationally difficult to simultaneously use both the structured output learning and the kernel method. Our method avoids this difficulty by transforming the kernel function, and enables the kernelized structured output learning. We theoretically discuss the computational complexity of the proposed method and also empirically show its high efficiency and effectiveness through experiments in the task of identifying agreement and disagreement relations between utterances in meetings. Identifying agreement and disagreement relations consists of two mutually-correlated problems: identification of the utterance which each utterance is intended for, and classification of each utterance into approval, disapproval or others. We simultaneously use both of the structured output learning and the kernel method in order to take into account this correlation of the two problems.

Keywords

Structured Output Learning, Machine Learning, Passive-Aggressive Algorithm, Kernel, Meeting Records, Dialog Act, Adjacency-Pairs.

1 Introduction

Structured output learning is a method that learns a model in order to predict the structured label of an instance. Typically, the structure is complex, and the set of possible labels is very large. Examples of structured labels are graphs, trees and sequences. Recently, for the algorithms of structured output learning, Support Vector Machine (Thochantaridis, 04) and Passive Aggressive Algorithm (Crammer, 06) were expanded. Structured output learning plays an important role in natural language processing (NLP), including parsing and sequential role labeling.

In NLP, the polynomial kernel has proven effective because it can take into account the interaction between features. For example, for complicated tasks like dependency structure analysis, we need to consider a combination of features (Kudo, 00). The use of a kernel is necessary for non-linear classification, as well as for improvement of performance. However, structured output learning is hardly ever used with a kernel, mainly because evaluation of all possible labels by kernels would be necessary, and is computationally prohibitive. We construct a kernelized classifier that identifies agreement/disagreement relations between utterances in dialogue. Identifying agreement and disagreement relations consists of two mutually-correlated problems: identification of the utterance which each utterance is intended for, and classification of each utterance into approval, disapproval or others. This problem can be seen as the problem of finding edges between pairs of nodes, where the number of nodes is fixed, and it is equivalent to restricting the possible structure output in the output of structured output learning. We can apply our method to the problems that can be seen as identification of a graph with a fixed number of nodes.

Furthermore, when using structured output learning for complex NLP tasks, predicting a structure might include a number of different problems. In order to simultaneously learn a number of different problems as elements of a structured label, we define cost functions in consideration of the case where the proportion of classes differs among the problems.

In this paper, we propose a method that transforms the kernel to reduce the computational complexity of learning with restricted structured output and kernels, and propose cost functions to take into account the different class proportions among the problems, in order to apply structured output learning to a larger area of application. We evaluate our method on the task of identifying agreement and disagreement relations in the MRDA corpus (Shriberg, 04). On the large set of labels, experiments show that our method is exponentially faster than the conventional methods.

2 Related work

In structured output learning, we need to find the best label from an exponentially large set of labels in order to predict the label. Therefore, the complexity of predicting the label determines whether the problem can be solved in practical time or not. Specialized algorithms for each problem to improve efficiency are thus used for predicting a label. For example, if the label is a sequence, the Viterbi algorithm might be useful (Sittichai, 08), and if the label is a parse tree, parsing algorithms like CKY can be used to search for the best label (McDonald, 05). In general, these problems are learned by linear models.

When training a kernel-based model, the increasing number of support vectors has a bad influence on the complexity. For this reason, Orabona (2008) proposed a method that approximates a new vector through the existing support vectors in order to reduce the overall number of obtained support vectors. Similarly, the complexity of classification increases according to the increase of the set of support vectors in the Support Vector Machine (SVM). Keerthi (2006) proposed a method to approximate the size of the support vector set.

Another approach to overcome the increase of the support vectors is a technique that expands the kernel so that it avoids dealing with the support vectors (Moh, 08). This method expands a polynomial kernel in order to treat the induced feature space as the feature vector in linear models. However, this is infeasible for a large feature set.

3 Passive Aggressive Algorithm

Passive Aggressive Algorithm (Crammer, 06) is a family of perceptron-like online max margin algorithms. It has linear complexity in the number of examples. Therefore, this algorithm is faster than batch-algorithms such as SVM, and requires less memory. Crammer proposed an expansion to structured output learning, and a derivation algorithm for learning with a kernel. At each step, this algorithm conservatively updates the model so that it can correctly classify the misclassified instance $\mathbf{x}$. For details, refer to (Crammer, 06).

Algorithm 1 shows the expansion to structured output learning with kernel. Each instance $\mathbf{x}^{(i)}$ is paired with a correct label $\mathbf{y}^{(i)}$. We call the pair $(\mathbf{x}, \mathbf{y})$ added to the model $\mathcal{W}$ a *support vector*, such as in SVM, and τ is the weight of the support vector. So the model $\mathcal{W}$ is a set of tuples:(τ , $\mathbf{x}$, $\mathbf{y}$). Each pair of an instance and a label corresponds to a feature vector that is given by $\Phi(\mathbf{x}^{(i)}, \mathbf{y})$. The prediction problem is reduced to finding the best label $\hat{\mathbf{y}}$ from the possible labels $\mathcal{Y}$:

$$\bar{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \sum_{\{\tau^{(t)},\mathbf{x}^{(t)},\mathbf{y}^{(t)}\}\in\mathcal{W}} \tau^{(t)} K(\Phi(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}), \Phi(\mathbf{x}^{(i)}, \mathbf{y}))$$

To assign a different cost for each misclassified instance during the learning, a cost function $\rho(\mathbf{y}, \mathbf{y}')$ is introduced, associated with every pair of correct label $\mathbf{y}$ and predicted label $\hat{\mathbf{y}}$. We assume that $\rho(\mathbf{y}, \mathbf{y}') = 0$ if $\mathbf{y}' = \mathbf{y}$ and that $\rho(\mathbf{y}, \mathbf{y}') \geq 0$ whenever $\mathbf{y} \neq \mathbf{y}'$. At each update step, the algorithm updates the model so that the following constraint is going to be satisfied,

$$\mathcal{W}_{\text{updated}} = \mathcal{W} \cup (\tau, \mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \cup (-\tau, \mathbf{x}^{(i)}, \bar{\mathbf{y}}).$$

At step 4, the algorithm maximizes the following expression by finding the label that violates this constraint to the highest extent,

$$\bar{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \sum_{\{\tau^{(t)},\mathbf{x}^{(t)},\mathbf{y}^{(t)}\}\in\mathcal{W}} \tau^{(t)} K(\Phi(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}), \Phi(\mathbf{x}^{(i)}, \mathbf{y})) + \sqrt{\rho(\mathbf{y}^{(i)}, \mathbf{y})}. \quad (1)$$

At steps 5 and 6, unless there is a sufficient margin between the example with the correct label and the other examples, we calculate the weight τ. τ is a real number that ensures necessary margin between the example with the correct label and the example with the resultant label $\bar{\mathbf{y}}$. Step 7 updates the model by τ and $\bar{\mathbf{y}}$ as follows:

$$\mathcal{W}_{\text{updated}} = \mathcal{W} \cup (\tau, \mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \cup (-\tau, \mathbf{x}^{(i)}, \bar{\mathbf{y}}).$$

There are two problems with simultaneous use of structured output and polynomial kernel. One is clear from formula (1). Following the increase in the number of support vectors, the computational complexity increases compared to a linear model. Since a maximization (1) is carried out at each iteration, the computational complexity increases not only during classification, but also during learning.

Algorithm 1 Passive Aggressive Algorithm

```
Input: $S = ((\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), ..., (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})), C$
 1: initialize model
 2: for $iteration = 1, 2, \ldots$ do
 3:   for $i = 1, ..., N$ do
 4:     get most violated label $\bar{\mathbf{y}}$
 5:     if $(\bar{\mathbf{y}} \neq \mathbf{y}^{(i)})$ then
 6:       calculate $\tau$ from $\rho(\mathbf{y}^{(i)}, \bar{\mathbf{y}})$
 7:       update model
 8:     end if
 9:   end for
10: end for
11: return model
```

We should also notice that the number of support vectors tends to be large in online learning. In addition, since structured output learning has as many examples as the number of pairs of an instance and a label, the number of support vectors tends to be even larger.

The second problem is that since the algorithm classifies the instances in the kernel space, we cannot evaluate the intermediate scores corresponding to the elements of label structures and their features. Therefore, most of the available conventional decoding algorithms such as Viterbi cannot be used. Moreover, since step 4 requires the maximization of the sum of more than one polynomial kernel function, it is difficult to solve this problem efficiently.

4 Formal definition

In this section, we give a formal definition of the problem of applying the polynomial kernel to structured output prediction.

- label $\mathbf{y}$

  In this paper, we assume that the label is a binary vector of length m. Most of the data structures, such as graphs or sequences, can be represented by binary vectors.

- instance $\mathbf{x}$

  $\mathbf{x}$ denotes the vector representing an instance.

- Φ function

  In algorithm 1, the Φ function generates a feature vector from a label and an instance. $\oplus$ denotes an operator that concatenates two vectors, m denotes the label length, and y_i denotes a label element. We define the Φ function as follows,

  $$\Phi(\mathbf{y}, \mathbf{x}) = (y_1\mathbf{x}) \oplus (y_2\mathbf{x}) \oplus \cdots \oplus (y_m\mathbf{x}).$$

  For example, for a given pair $\mathbf{y} = (1, 0, 1, 0), \mathbf{x} = (1, 2, 3, 4)$, we obtain $\Phi(\mathbf{y}, \mathbf{x}) = (1, 2, 3, 4,\ 0, 0, 0, 0,\ 1, 2, 3, 4,\ 0, 0, 0, 0)$. By combining each of the label elements with the vectors that represent the instances, this Φ function generates a range of features in a vector corresponding to each label element. That is, the weights of each feature can be determined for each label element. We can decide whether the feature is a positive evidence or a negative evidence for each label element.

- cost function

  We define three cost functions for the labels, each one of which is represented as a binary vector. Section 6 explains the cost functions in detail.

5 Transformation of the polynomial kernel

As a kernelized learner, we often use a polynomial kernel in NLP in order to treat the combination of features such as words and dependencies. Let m be the label length, and $K(\mathbf{v}, \mathbf{v}')$ be a p degree polynomial kernel. We transform this kernel to reduce the computational complexity. The transformation is composed of integrating the Φ function and polynomial kernel, and decomposing the integrated kernel.

5.1 Integrating the Φ function and the kernel

Let us first consider reducing the computational complexity per kernel function evaluation. For calculating the kernel, we must regenerate the feature vector from $\mathbf{x}$ and $\mathbf{y}$, or cache the kernel values. However it is impractical to hold all the $N2^m$ feature vectors. In addition, we must deal with all feature vectors at each iteration. The access to feature vectors does not have locality of reference. Therefore caching support vectors is not efficient. For this reason, we calculate the kernel value directly, by integrating the polynomial kernel and the Φ function.

Let us calculate the kernel between the vectors generated by the Φ function in due order. For simplicity, the constant term in the polynomial kernel is dropped out, without loss of generality. The p degree polynomial kernel can be expanded as follows:

$$\begin{aligned} & K(\Phi(\mathbf{y},\mathbf{x}), \Phi(\mathbf{y}',\mathbf{x}')) \\ = \ & ((y_1 \cdot \mathbf{x} \oplus y_2 \cdot \mathbf{x} \oplus \ldots \oplus y_m \cdot \mathbf{x}) \cdot \\ & (y'_1 \cdot \mathbf{x}' \oplus y'_2 \cdot \mathbf{x}' \oplus \ldots \oplus y'_m \cdot \mathbf{x}'))^p. \end{aligned}$$

Since in the kernel space an inner product can be obtained by the range of features corresponding to each label element, the above formula can be tranformed as follows:

$$K(\Phi(\mathbf{y},\mathbf{x}), \Phi(\mathbf{y}',\mathbf{x}')) = \{(y_1 \cdot y'_1)(\mathbf{x} \cdot \mathbf{x}') + \ldots + (y_m \cdot y'_m)(\mathbf{x} \cdot \mathbf{x}')\}^p,$$

and we can extract the products of the dot-product of instance vectors $(\mathbf{x} \cdot \mathbf{x}')$, $K(\Phi(\mathbf{y},\mathbf{x}), \Phi(\mathbf{y}',\mathbf{x}')) = ((\mathbf{y} \cdot \mathbf{y}')(\mathbf{x} \cdot \mathbf{x}'))^p$. Thus, a kernel can be represented by a dot-product of labels $(\mathbf{y} \cdot \mathbf{y}')$ and a dot-product of instance vectors $(\mathbf{x} \cdot \mathbf{x}')$. It turns out that we can evaluate the polynomial kernel without the Φ function. Here, let the kernel integrated with Φ function be denoted by K_{ex} as follows:

$$K_{ex}(\mathbf{y},\mathbf{x},\mathbf{y}',\mathbf{x}') = ((\mathbf{y} \cdot \mathbf{y}')(\mathbf{x} \cdot \mathbf{x}'))^p. \qquad (2)$$

Refer to Figure 1 for intuitive explanation. Through the kernel integrated with Φ, we can evaluate a kernel by only the dot-product between the instance vectors, regardless of the label length.

If we do not integrate Φ with the kernel, evaluation of the kernel costs $(\text{label length}) \times (\text{feature size})$ computational time and memory. By contrast, evaluation costs only $(\text{label length}) + (\text{feature size})$ for both in our calculation. Since we have to only cache the kernel between the instances, cache efficiency increases significantly.

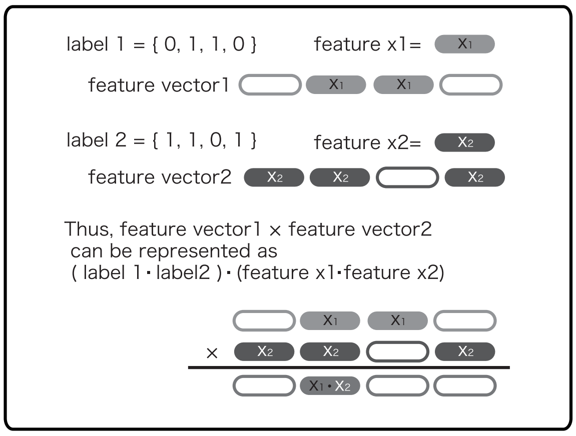

Fig. 1: *Integrating Φ function and kernel*

5.2 Decomposing the kernel

In order to reduce the number of calls to the kernel function as much as possible, we expand the integrated kernel further, limiting ourselves to the case of second degree polynomial kernels for the sake of simplicity.

In the Passive Aggressive Algorithm with a kernel, the number of suppor vectors increases during the iterations, and becomes an arbitrarily large set. For this problem, Moh (2008) proposed a method that expands a second degree polynomial kernel to an induced feature space, and treated it as a linear model to avoid treating support vectors for memory complexity. However, since Moh's method must treat a large space that is of a square of the size of a feature set, this has the opposite effect that the computational space is increasing. Thus, this method cannot treat a large number of features. If we expand the kernel in a feature space, it cannot benefit from the kernel trick and it must treat a large feature space as in Moh(2008). We expand the kernel not only in the feature space, but also in the label space. Therefore, we can expand the kernel efficiently, if $\text{label length} < \text{feature size}$.

K_{ex} can be decomposed and represented as a combination of y_i that belongs to $\mathbf{y}$, where the constant term in a polynomial kernel is dropped out without loss of generality,

$$K_{ex}(\mathbf{y},\mathbf{x},\mathbf{y}',\mathbf{x}') = ((\mathbf{y} \cdot \mathbf{y}')(\mathbf{x} \cdot \mathbf{x}'))^2 = (\mathbf{y})^2 \left((\mathbf{y}')^2 (\mathbf{x} \cdot \mathbf{x}')^2\right).$$

Here, we consider the calculation of the kernel score $S(\mathbf{y},\mathbf{x})$ between an example $(\mathbf{y},\mathbf{x})$ and the support vectors $\{\tau^{(t)}, \mathbf{y}^{(t)}, \mathbf{x}^{(t)}\}$. We decompose $\mathbf{y}$ and $\mathbf{y}^{(t)}$ to each y_i and $y_i^{(t)}$, and expand the square. So let γ_{ij} denote $\sum_{\{t|(\tau^{(t)},\mathbf{y}^{(t)},\mathbf{x}^{(t)}) \in \mathcal{W}\}} \tau^{(t)} y_i^{(t)} y_j^{(t)} (\mathbf{x} \cdot \mathbf{x}^{(t)})^2$, because γ_{ij} is constant in terms of $\mathbf{y}$. We then obtain the score of the example:

$$\begin{aligned} S(\mathbf{y},\mathbf{x}) &= \sum_{(\tau^{(t)},\mathbf{y}^{(t)},\mathbf{x}^{(t)}) \in \mathcal{W}} \tau^{(t)} K_{ex}(\mathbf{y},\mathbf{x},\mathbf{y}^{(t)},\mathbf{x}^{(t)}) \\ &= \sum_{\{i,j|i \neq j,\ i,j<m\}} y_i y_j \gamma_{ij} + \sum_{\{i|i \leq m\}} y_i^2 \gamma_{ii}. \end{aligned}$$

As shown above, support vectors can box in the parameters $\gamma \in \mathcal{R}^{m^2}$. For the same instance vector $\mathbf{x}$ and the same model, we can calculate the score using γ for each $\mathbf{y}$ without calls to the kernel. We call the expanded kernel a “transformed kernel”.

5.3 Predicting with polynomial kernelization

At step 4 of Algorithm 1, we solve the maximization problem. We exploit the expansion so that we obtain the label which violates the constraints to the highest extent.

Algorithm 2 predicts $\bar{\mathbf{y}}$ using the transformed kernel. $\mathbf{1}$ denotes the label, each element of which is 1. $\mathbf{e}_i$ denotes a label that is the vector with a 1 in the i-th element and 0 elsewhere. And $\tau^{(k)}$ denotes the weight on the k-th support vector. $\mathbf{sv}_{ki} = \Phi(\mathbf{e}_i, \mathbf{SV}_k)$ denotes the feature vector generated by Φ from the k-th support vector and $\mathbf{e}_i$. We henceforth define $y^{(0)}$ as always equal to 1, so that we calculate γ with a constant term, and the label whose length is m implicitly includes the constant term $y^{(0)}$.

```
Algorithm 2 for finding $\bar{\mathbf{y}}$.
Input: $\mathbf{x} = \Phi(\mathbf{1}, \mathbf{x}), \tau, \mathcal{W} = \{\mathbf{sv}_1, ..., \mathbf{sv}_n\}$
Input: true_y //correct label(in training only)
1: for all $\{(i,j) | 0 < i \le j \le m\}$ do
2:     $\gamma_{ij} = \sum_{\mathbf{sv}_k \in \mathcal{W}} \tau^{(t)} \beta_{ij} K(\mathbf{sv}_{ki}, \mathbf{x}) K(\mathbf{sv}_{kj}, \mathbf{x})$
       $\beta_{ij} = \begin{cases} 1 & if (i = j) \\ 2 & otherwise \end{cases}$
3: end for
4: for $0 \le i \le m$ do
5:     $\gamma_{0i} = \gamma_{i0} = \sum_{\mathbf{sv}_k \in \mathcal{W}} \tau^{(t)} K(\mathbf{sv}_{ki}, \mathbf{x})$
       // processing constant terms.
6: end for
7: $\bar{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \{0,1\}^m} S(\mathbf{y}, \mathbf{x})$ //when classifying
   $\bar{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \{0,1\}^m} S(\mathbf{y}, \mathbf{x}) + \rho(\mathbf{true\_y}, \mathbf{y})$ //when training
8: return $(\bar{\mathbf{y}})$
```

Calculating γ requires calling the kernel function $n(m+1)^2$ times, but the evaluation of each label requires only the calculation of the polynomial expression whose coefficient is γ. Thus, even the evaluation of all possible labels has only to call the kernel $n(m+1)^2$ times.

Additionally, the Passive Aggressive Algorithm trains the model incrementally, and the weight of the support vector added to the model does not change. In this way, if we hold γ for all instances, γ can be updated according to the support vectors newly added to the model. Thus, complexity can be reduced even further.

5.4 Discussion

Here, we discuss the computational complexity of the learning. Let N be the number of examples, and let h be the number of support vectors included in the model at a given point. Let H be the number of the final support vectors, I be the number of iterations needed to obtain H support vectors, and m be the label length.

We assume that the misclassification rate of training data is constant while training the model. We then need to perform classification calculation $\lambda = \frac{N\,I}{H}$ times in order to obtain one support vector. In the following, we will see the number of calls to the kernel required to obtain all the support vectors for each case of without transformation and with transformation.

- Without kernel transformation

  Since evaluation of each example requires $h2^m$ calls to the kernel function, obtaining one support vector requires $h2^m\lambda$ calls. Thus, the number of calls to the kernel to obtain H support vectors is, $\sum_{h=1}^{H} h\lambda 2^m = NI(H+1)2^{m-1}$. In practice, misclassifications decrease in number with training and one support vector requires more classification examples. Hence complexity can become larger.

- When transforming the kernel

  Since evaluation of the examples requires only the calculation of the kernel of the new support vectors, it needs $\frac{H(m+1)^2}{I}$ calls to the kernel function on average. Therefore, the number of calls to the kernel to get H support vectors is, $H\frac{H(m+1)^2}{I}\lambda = NH(m+1)^2$.

In the cases where the label length is long or training needs many iterations, the computational complexity benefits from the transformation of the kernel.

6 Cost function

In structured output learning, for a given pair of labels, a cost is calculated by a cost function. Hereby, we can introduce a “near error” and a “distant error”, and impose a little penalty to near error and a large penalty to distant error. We define three cost functions. Each cost function defines what is ”near” and ”distant”. s is a scale parameter in the following.

- 0/1 cost

  $\rho_{0/1}(\mathbf{y}, \mathbf{y}') = \begin{cases} 0 & if\,\mathbf{y} = \mathbf{y}' \\ s & otherwise \end{cases}$

  It returns s if the labels differ, and 0 otherwise. It is the most basic cost function that can be defined for general labels.

- average cost

  $\rho_{average}(\mathbf{y}, \mathbf{y}') = \frac{1}{m}\sum_{i=1}^{m} \begin{cases} 0 & if\, y^{(i)} = y'^{(i)} \\ s & otherwise \end{cases}$

  It returns $s \times$ (the number of different elements in the label) divided by the label length. It means that errors in several label elements induce a larger penalty.

- Asymmetric cost

  $$\rho_{asm}(\mathbf{y}, \mathbf{y}') = \frac{1}{m}\sum_{i=1}^{m} \begin{cases} 0 & y^{(i)} = y'^{(i)} \\ s \cdot j_i & y^{(i)} = 1, y^{(i)} \neq y'^{(i)} \\ s & otherwise \end{cases}$$

  This cost function returns $s \cdot j_i$ if a positive element is incorrectly classified as a negative element. If there is a bias between positive and negative label elements, the model will learn disproportionally by rote, because of which it gains a rather oversized margin, and the rest cannot gain sufficient margin. In order to remedy the bias and gain decent margins, an asymmetric cost function gives a different cost when the positive or negative elements are mistaken. It changes the width of the margin that must be

Table 1: *Dialogue example*

A	wait, is this a computer science conference ?
A	or is it a
B	um, well, it's more . . .
B	it's both right.
B	it's it's sort of t- cognitive neural psycho linguistic
B	but all for the sake of doing computer science
B	so it's sort of cognitive psycho neural plausibly motivated architectures of natural language processing
B	so it seems pretty interdisciplinary

reserved. We assign different parameters j_i to each label, so that this function can absorb the positive and negative bias that is different for each element.

7 Experiments

We examine the task of identifying agreement and disagreement between utterances to verify the efficiency and the effectiveness of our method. Identifying agreement and disagreement between utterances is to predict whether each utterance shows agreement or disagreement, and inter-utterances have a link.

7.1 Data

We used the MRDA corpus that has been used by related works (Galley, 2004). This corpus contains dictated text and audio data collected from 75 multi-party meetings in ICSI. The meetings, one hour duration each, have been held on a weekly basis by 6.5 researchers on average. For all utterances in this corpus, annotators labeled that the Dialog Acts, speakers, Adjacency-Pairs, etc.

Each Dialog Act is a category of utterances defined according to their intent. There are 44 Dialog Acts. Among them, we regard 4 tags, Acknowledge-answer("bk"), Accept("aa"), Accept-part("aap"), Maybe("am"), as agreement. We regard other 2 tags, Reject("ar"), Reject-part("arp") as disagreement. Adjacency-Pairs are another kind of tags. We regard an utterance pair is linked if they are annotated with the Adjacency-Pairs tag. We show the dialogue example in Table 1.

7.2 Experimental settings

In this experiment, we aim to predict the agreement and disagreement relations between the utterances, segmented into groups of 3 continuous utterances: first, second, and third utterances. There are 3 possible links. For each link, there are 3 possible values: agreement, disagreement, others. Therefore, the label length m is 9. In this paper, we train and classify shifting the segments by 1 utterance. That is, the second utterance from an example is the first utterance on the next example. We also used as features the preceeding and succeeding 3 utterances, 7 utterance in all, to classify on our method. The features that denote the content of utterances are the word length, uni-grams, bi-grams, tri-grams, head 2 words, and tail 2 words. The features that denote relations between utterances are whether the speaker is the same or not, and the time interval. We cannot evaluate this problem precisely by accuracy, since the classes are biased in size. Thus, we used F-value in order to evaluate. We used 12499 examples to train, and 9200 examples for testing. We do not determine an iteration limit in the Passive Aggressive Algorithm; instead, we used the model of the point of convergence. The second degree polynomial kernel was used, and the constant term is set to 1.

7.3 Execution time

We also measure the execution time. When the kernel is not integrated, the execution time tends to be prohibitively long. So we experimented with a small training dataset in that case.

Additionally, we used fixed parameters, because parameters influence the execution time. The number of iterations is 15. We used the average cost. We set the scale factor of the cost function s to 10. We used the second degree polynomial kernel in the evaluations of both the transformed and non-transformed kernels.

8 Results

8.1 Effect of the cost function

We show the result of our examination of the performance of the cost function in Table 2. We first compare the result with the zero-one cost with the result with the average cost. When the zero-one cost is used, the utterance position changes the result significantly, but classifying each position utterance is the same problem.

On the other hand, when the average cost is used, the utterance position does not change results much. Since the zero-one cost judges only the overall correctness of the predicted relations, it learns to reserve margin against both near errors and distant errors. As a result, it reserves oversized margin against mistakes of the label elements during learning. But, when the average cost is used, the mistakes of label elements change the score; So it learns to reserve an appropriate margin against elements of each label.

Furthermore, when the asymmetric cost is used, the method performs well for all label elements. This is because this asymmetric cost absorbs the proportion of positive contents and negative contents, by implementing differential penalty for the mistakes. We show the effect of the parameters of asymmetric cost in the next section.

Table 2: *Performance of the cost function*

Cost function	0/1	Average	Asymmetric ($\mathbf{j}$=3)
Agreement (utterance 1)	0.337	0.394	0.490
Agreement (utterance 2)	0.170	0.343	0.462
Agreement (utterance 3)	0.097	0.237	0.414
Disagreement (utterance 1)	0.016	0.000	0.134
Disagreement (utterance 2)	0.000	0.032	0.032
Disagreement (utterance 3)	0.000	0.000	0.076
Link(utterance 1-2)	0.030	0.074	0.305
Link(utterance 2-3)	0.021	0.088	0.366
Link(utterance 1-3)	0.012	0.083	0.277

8.2 Effect of the parameters j

We show the results with different values of parameters $\mathbf{j} \in \{j_i\}^m$ in Table 3. We can change the penalty that is given when the positive content is incorrectly classified, by changing $\mathbf{j}$. We observe an improvement in the classification of agreements or links, and particularly in the classification of disagreements where these are few positive examples. Increasing $\mathbf{j}$ does not improve the performance unlimitedly. The optimal values for $\mathbf{j}$ can be determined from the proportion of the positive and negative examples, and we must fit $\mathbf{j}$ to the corresponding problem.

Table 3: *Influence of* **j**

j	0.5	1	10	50	100
Agreement (1)	0.290	0.382	**0.526**	0.500	0.479
Agreement (2)	0.226	0.327	**0.510**	0.497	0.460
Agreement (3)	0.160	0.216	**0.514**	0.507	0.464
Disagreement (1)	0.000	0.000	0.245	**0.320**	0.287
Disagreement (2)	0.000	0.000	0.201	**0.306**	0.277
Disagreement (3)	0.000	0.000	0.258	**0.320**	0.316
Link (1-2)	0.173	0.088	0.472	**0.508**	0.445
Link (2-3)	0.038	0.142	**0.529**	0.518	0.456
Link (1-3)	0.027	0.069	0.431	**0.450**	0.413

Table 4: *Comparison with linear model*

Model	Kernelized	Linear
Agreement	**0.512**	0.501
Disagreement	**0.316**	0.263
Link(utterance 2-3)	**0.550**	0.494
Link(utterance 1-3)	**0.454**	0.418

8.3 Effect of the kernel

Based on the results above, we optimized the weight **j** for agreement/disagreement and link, and compared with the linear model. The weight parameters are as follows: $j_{yea} = 3$, $j_{nay} = 10$, $j_{link_{1-2,2-3}} = 6$, $j_{link_{1-3}} = 5$. We set the scale factor of the cost function s to 5. In linear model: $j_{yea} = 3$, $j_{nay} = 100$, $j_{link_{1-2,2-3}} = 6$, $j_{link_{1-3}} = 5$. We set the scale factor of linear model s_{linear} to 4. We chose the parameters for test data. Thus, the results are the upper limit that we can obtain by tuning the parameters.

We show the results with this settings in Table 4. The linear model cannot deal with corresponding words among the utterances because it cannot treat a combination of features, and slows down the performance, especially when classifying links. In classification of agreements/disagreements, the polynomial kernel improves the performance, too.

8.4 Computational complexity

We measured the execution times and compared them in Table 5. For both cases of using transformation or not, the execution time is proportional to $(\text{example size}) \times (\text{support vector size})$.

Without overheads, the difference of these execution times is close to the theoretical complexity difference when the kernel is called $2^m = 512$ times and the complexity for each kernel is $m = 9$ times higher, coming together as in total 4608 times.

Table 5: *Execution time*

Number of training examples	50	100	150
Support vectors	118	183	294
Using non-transformed kernel	15523s	56227s	139590s
Using transformed kernel	3. 19s	8.14s	28.65s
Ratio	×4866	×6907	×4872

9 Conclusion

In this paper, we proposed the cost functions to take into account the different class proportions between the problems, and a method that transforms the kernel to reduce the computational complexity of learning with structured output and kernels. This algorithm is based on one of the online max margin algorithm, Passive Aggressive Algorithm, so it learns fast and uses a small amount of memory. We evaluated our method on the task of identifying agreement and disagreement relations, and we empirically and theoretically showed the computational complexity of the proposed method, and also the efficiency of using a polynomial kernel for structured output learning.

References

[1] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, Yoram Singer, Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, Vol. 7, pp. 551–585, 2006.

[2] Michel Galley, Kathleen McKeown, Julia Hirschberg, Elizabeth Shriberg, Identifying agreement and disagreement in conversational speech: Use of bayesian networks to model pragmatic dependencies. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pp. 669–676, 2004.

[3] Taku Kudo, Yuji Matsumoto, Japanese dependency structure analysis based on support vector machines. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pp. 18–25, 2000.

[4] Elizabeth Shriberg, Raj Dhillon, Sonali Bhagat, Jeremy Ang, and Hannah Carvey, The ICSI meeting recorder dialog act (MRDA) corpus. In *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue*, pp. 97–100, 2004.

[5] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, Yasemin Altun, Support Vector Learning for Interdependent and Structured Output Spaces. In *Proceedings of the 21st International Conference on Machine Learning*, pp. 823–830, 2004.

[6] Yasemin Altun, Ioannis Tsochantaridis, Thomas Hofmann, Hidden Markov Support Vector Machines. In *Proceedings of the 20th International Conference on Machine Learning*, pp. 3–10, 2003.

[7] Yvonne Moh,Thorsten, Joachim Buhmann, Kernel Expansion for Online Preference Tracking. In *proceedings of The International Society for Music Information Retrieval*, pp. 167–172, 2008.

[8] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, Yasemin Altun, Support Vector Learning for independent and Structured Output Spaces. In *Proceedings of the 21st International Conference on Machine Learning* , p. 104 , 2004.

[9] Francesco Orabonal, Joseph Keshet, Barbara Caputo, The projectron: a bounded kernel-based Perceptron. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 720–727 , 2008.

[10] Jiampojamarn Sittichai, Cherry Colin, Kondrak Grzegorz, Joint Processing and Discriminative Training for Letter-to-Phoneme Conversion. In Proceedings of 2008 Annual Meeting on Association for Compurational Linguistics and Human Language Technology Conference, pp. 905–913, 2008.

[11] S. Sathiya Keerthi, Olivier Chapelle, Dennis DeCoste, Building Support Vector Machines with Reduced Classifier Complexity. *The Journal of Machine Learning Research*, Volume 7, pp. 1493–1515, 2006.

[12] Ryan McDonald, Koby Crammer, Fernando Pereira, Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pp. 91–98 , 2005.