NAIST-IS-DD0461007

Doctoral Dissertation

Discriminative Learning Methods for Interdependent Decision Problems in Natural Language Processing

Hideto Kazawa

September 20, 2006

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Hideto Kazawa

Thesis Committee:

Yuji Matsumoto, Professor	(Chairperson)
Hiroyuki Seki, Professor	(Member)
Shin Ishii, Professor	(Member)
Kentaro Inui, Associate Professor	(Member)

Discriminative Learning Methods for Interdependent Decision Problems in Natural Language Processing*

Hideto Kazawa

Abstract

Many natural language processing (NLP) tasks involve multiple inputs and/or outputs, which are strongly correlated to each other. We call such problems "interdependent decision problems (IDPs)" since the decisions on outputs should be interdependent to each other. In this thesis, we focus on three typical IDPs in NLP and investigate discriminative learning methods for these problems.

The first IDP is **sentence selection**, which is the process of selecting sentences from a document according to some criterion. Sentence selection is an IDP in the sense that whether an item is selected depends on other items in the input set and cannot be decided only from the features of the item. In Chapter 2, we proved that selection problem can be converted into classification problem. Then a new learning algorithm, Selection SVM, is proposed to solve successive selection problems. Experimental results on an artificial dataset and a sentence selection dataset are also reported.

The second IDP is **multi-topic text categorization**, which is a labeling process of assigning all (possibly multiple) relevant topics to a text. Multi-topic text categorization is an IDP decision problem since topics often show strong correlation among them. In Chapter 3, we address the problem of multi-topic text categorization. First We propose a new learning algorithm, Maximal Margin Labeling (MML), and also describe efficient algorithms for MML. MML is tested on datasets of Web pages, and the results are reported.

The third IDP is **sequence tagging**, which is a process of assigning a tag from given tag set to each word in a sequence. Sequence tagging can be called

*Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0461007, September 20, 2006.

a structured IDP since it has a chain structure on outputs, which naturally suggests that the correlations with near outputs are much stronger than those with far outputs. In Chapter 4, we presented an efficient algorithm for computing an optimal tagging order. Then we proposed a learning algorithm of strategy function, which predicts best tagging position. The experiments using a real sequence tagging data are described.

The future works related to the above problems are also discussed.

Keywords:

natural language processing, sentence selection, multi-topic categorization, sequence tagging, machine learning, discriminative model

Acknowledgements

This thesis is based on the studies I have done as a member of Knowledge Processing Research Group of NTT Communication Science Laboratories. I would like to thank all the members of the group, especially Eisaku Maeda, Hideki Isozaki, Tsutomu Hirao, Hirotoshi Taira and Tomonori Izumitani for producing ideas for the studies. Their inputs were critical and greatly contributed to improve the quality of my research.

The PMM's programs and the multi-topic datasets I used were kindly provided by Naonori Ueda and Kazumi Saito of Nippon Telegraph and Telephone Corporation, and Yuji Kaneda of NTT Resonant Inc. I would like to thank them, too.

I would also like to thank Prof. Kazumitsu Matsuzawa of Kanagawa University and Prof. Tsuneaki Kato of The University of Tokyo, who were group leaders when I started the career as a researcher in NTT. Without their helps and encouragements, the works in this thesis could not have even started.

At last many thanks go to Prof. Yuji Matsumoto and the members of NAIST Matsumoto Lab. The discussions and informal chats with them always inspire me and make my research life more joyful one.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1. Natural Language Processing and Discriminative Learning

In recent years, the design of natural language processing (NLP) algorithms has experienced a drastic paradigm shift from an artistic fine-tuning of elaborate hand-crafted heuristics to an automatic estimation of naive but massive model parameters. From a social viewpoint, this is because the cost of computation has been constantly declining and a lot of electronic linguistic data has become available due to the spread of personal computers into every aspect of our life. On the other hand, this is also because NLP researchers came to realize that language is such a complex object that we need more complex models and principled methodologies to adjust them [1].

The parameter estimation is usually done by *machine learning* algorithms. Here we use "machine learning" in a broad sense to refer to any method which estimates model parameters from given *training data*. Training data can be simply a set of instances of an interesting linguistic phenomenon or a set of input-output pairs of a specific NLP task. Given training data, machine learning algorithms estimate model parameters by searching parameters which minimize (or maximize) an objective function of learning.

Corresponding to what is modeled, there are two types of machine learning. Models of the first type methods are probability distributions which give probabilities that target linguistic *phenomena* happen. On the other hand, models of

[1] You can find a similar argument in the preface of [1]

the second type are deterministic functions whose values are outputs of target natural language *processing*. The first models are called "generative models", and the second ones "discriminative models".

Generative models are more general than discriminative models since a process with input X and output Y can be modeled as a phenomenon whose joint probability is $\Pr(X, Y)$, and a discriminative model can be derived from the joint probability by $\arg\max_Y \Pr(Y|X)$. Discriminative models, however, are more specific to target tasks and then are likely to become much simpler than generative models. Generally speaking, parameters of simple models can be accurately estimated and this is why discriminative models often show better performance on various NLP tasks than generative models. The model simpleness also implies that we need only a small number of training data, which is another advantage for discriminative models since the cost of training data construction is usually large in NLP. These advantages make discriminative-model-based machine learning (*discriminative learning*) quite popular in NLP.

2. Interdependent Decision Problems

NLP tasks often involve multiple inputs and/or multiple outputs. For example, part-of-speech(POS) tagging, whose goal is to assign POS to words in a sentence, involves multiple inputs (words) and multiple outputs (part-of-speeches). It is also often the case that such multiple inputs and/or outputs are strongly correlated. For the case of POS tagging, each POS is strongly correlated to neighbor POS: for example, verbs do not appear after articles. We call the problems which involves (strongly) correlated inputs and/or outputs "interdependent decision problems" since the decisions on inputs/outputs are interdependent to each other.

For interdependent decision problems, it is natural to expect that machine learning methods can benefit from incorporating the input/output correlations into the models.

For generative models, however, such incorporation is usually difficult. When modeling joint probability distributions, one usually assume some dependencies between input and output variables. In many cases, these dependencies are designed to reflect causal relationships between variables. However, it is often the case that to capture the causality in linguistic phenomena is very difficult. For example, it is far from obvious whether the POS of a word is a cause or a result

of the previous word's POS.

For discriminative models, situations are quite different since we do not have to worry about things like what causes what. We can design models more freely since what we need is a model of process, not phenomenon, and any algorithm can be a model of process as long as its input and output meet the problem's specification.

3. Problems Addressed in This Thesis

In this thesis, we will address the following problems as typical examples of interdependent decision problems.

3.1 Sentence Selection: an Interdependent Input Problem

The goal of sentence selection is to select sentences from a document (i.e, a set of sentences) according to some criterion. Sentence selection is considered to be an important step in automatic text summarization [2]. It can be used as a preprocessor before more elaborated operations such as sentence compaction, or as a stand-alone process to generate extracts.

Discriminative learning of selection (*selection learning*) is a learning task in which the learner attempts to estimate the selection criterion from samples of the target selection process. More formally speaking, the learner is given samples of input item (i.e, sentence in sentence selection) sets S and output item sets $T \subset S$, and then searches a set-to-set function F which approximates well T by $F(S)$ even for unseen S.

Sentence selection is an example of interdependent decision problem in the sense that whether an item is selected depends on other items in the input set and cannot be decided only from the features of the item. Since the dependency mainly exists among input items, we say that selection learning is an interdependent *input* problem.

3.2 Multi-topic Text Categorization: an Interdependent Output Problem

Multi-topic text categorization is a labeling task whose goal is to assign all relevant topics to a text. Topics should be selected from a given topic set and a

single text can have multiple topics. There are many examples of multi-topic text categorization in real world applications: news companies use multiple topic tags to classify articles [3]; search engines link multiple categories to a single web page for directory services [4].

Learning of multi-topic text categorization is a learning task in which the learner attempts to estimate the labeling rules from samples of text and assigned topics.

Multi-topic text categorization is an example of interdependent decision problem since the outputs (i.e, topics) often shows strong positive and negative correlation among them. Since the dependency mainly exists among outputs, we say that selection learning is an interdependent *output* problem.

3.3 Sequence Tagging: a Structured Output Problem

The goal of sequence tagging is to assign a tag from given tag set to each word in a sequence. (We use term *sequence* in the meaning of *word sequence* throughout this thesis.) Many important NLP problems such as token segmentation, part-of-speech tagging and shallow parsing can be regarded as sequence tagging problems.

The goal of learning of sequence tagging is to estimate tagging rules from samples of tagged sequences. In some tasks, samples are converted into a form of tagged sequence before learning. For example, in token segmentation task, whose goal is to segment a sequence into tokens a segmented sentence is converted into a character sequence in which each character has a tag indicating whether the character is at the beginning or end of a token. Using this kind of conversion, many NLP tasks can be solved as a sequence tagging task.

Sequence tagging is another example of interdependent output problems since neighbor outputs (i.e, tags) are usually strongly correlated. It also has a chain structure on outputs, which naturally suggests that the correlations with near outputs are much stronger than those with far outputs. Thus we say that sequence tagging is a structured output problem[2].

[2]Since the term "structure" implies dependencies, we omit "interdependent".

4. Main Contributions of This Thesis

4.1 Reduction of Selection Problem to Classification Problem

We proved that selection problem can be converted into classification problem, thus can be solved by any classifier learning algorithm. This is the first formal argument found in literatures, which justifies the use of classifier in selection learning problems. Then a new learning algorithm, Selection SVM, was proposed to solve successive selection problems. Experiments with an artificial dataset and a sentence selection dataset showed that Selection SVM performs well compared to other classifier-based methods and order learning methods.

4.2 Multi-class Classification Method for Multi-topic Categorization

We proposed a novel learning algorithm, Maximal Margin Labeling, which regards a set of multiple topics as a distinguished class and circumvents the difficulties caused by decomposition of multi-topic. Among related studies, this is the first full non-decompositional approach to multi-topic categorization problem. We also presented an approximation method in learning and efficient prediction algorithms to overcome the demanding computational cost of MML. In experiments on a collection of Web pages, MML outperformed other methods including SVM and showed better generalization.

4.3 Decision Order Learning in Sequence Tagging

First we presented an efficient algorithm for computing an optimal tagging (decision) order from given tagged sequence and tag classifiers and showed experimentally that even off-the-shelf classifiers can achieve an excellent tagging accuracy if they are led by a good decision order. Then we proposed a learning algorithm of strategy function, which predicts best position to tag during tagging. The experimental results showed that the learned strategies achieves much higher performance than non-learning strategies. The framework of strategy learning is novel and is not found in related literatures.

5. Organization of This Thesis

Chapter 2 addresses the problem of sentence selection. First we proved that selection problem can be converted into classification problem. Then a new learning algorithm, Selection SVM, is proposed to solve successive selection problems. Experimental results on an artificial dataset and a sentence selection dataset are also reported.

In Chapter 3, we address the problem of multi-topic text categorization. First we propose a new learning algorithm, Maximal Margin Labeling (MML), and also describe efficient algorithms for MML. MML is tested on datasets of Web pages, and the results are reported.

Chapter 4 addresses the problem of sequence tagging. First we presented an efficient algorithm for computing an optimal tagging order. Then we proposed a learning algorithm of strategy function, which predicts best tagging position. The experiments using a real sequence tagging data are described.

We summarize this thesis and discuss the directions of future work in Chapter 5.

Chapter 2

Sentence Selection

1. Introduction

The goal of sentence selection is to select sentences from a document (i.e, a set of sentences) according to some criterion. Sentence selection is considered to be an important step in automatic text summarization [2]. It can be used as a pre-processor before more elaborated operations such as sentence compaction, or as a stand-alone process to generate extracts.

Discriminative learning of selection (*selection learning*) is a learning task in which the learner attempts to estimate the selection criterion from samples of the target selection process. More formally speaking, the learner is given samples of input item (i.e, sentence in sentence selection) sets S and output item sets $T \subset S$, and then searches a set-to-set function F which approximates well T by $F(S)$ even for unseen S.

This chapter is organized as follows. First we briefly review related works in sentence selection and selection learning literatures in Section 2. Then the main problem of this chapter is stated in Section 3. In Section 4, we prove that selection problem can be converted into classification problem and a new learning algorithm, Selection SVM, is proposed to solve successive selection problems. In Section 5, experimental results on an artificial dataset and a sentence selection dataset are also reported. We summarize the results in Section 6.

2. Related Works

2.1 Classification

Hirao *et al.* applied Support Vector Machine (SVM) to sentence selection [5]. Their approach is as follows.

1. Regard selected sentences as positive examples and not-selected ones as negative examples.
2. Train SVM to discriminate these positive and negative examples.
3. Use the trained SVM as a scoring function to select sentences.

From a practical point of view, this approach has a potential advantage that one can apply any binary classifier for selection learning as long as the classifier gives a score of its prediction. It also showed good performance in experiments. However, it lacks theoretical justification of why classifier can be applied to selection problems in which items are NOT selected on the basis of their membership in some category.

2.2 Order Learning

In a sense, a selection process can be identified as a ranking process where the first selected item is first-ranked, the second selected item is second-ranked and so on. Under this identification, one can apply order learning methods to selection learning problems.

Order learning studies can be grouped into two types. The first group is categorical ordinal regression [6, 7, 5]. The second group is preference learning [8, 9, 10]. We will review these studies in turn.

Categorical Ordinal Regression

Categorical ordinal regression is a regression in which output (response) is a categorical rank, not a real value [6, 7, 5]. Categorical ranks are categories among which an order is defined. Thus categorical ordinal regression is essentially classification although it is called "regression".

On the other hand, ranks induced by a selection process are not categorical: the same item can be ranked differently by a selection process if input item sets

are different. Thus the application of categorical ordinal regression to selection problem also lacks a theoretical justification as the application of classification methods does.

Preference Learning

Preference learning is another classification-based method for order learning [10, 8, 9].

In preference learning, a classifier is trained to discriminate correctly ordered item pairs from wrongly ordered ones. (By "correctly ordered item pairs", we mean the pairs in which the first item is ranked higher than the second item.) At prediction, an optimal ranking is constructed from the predicted preferences [10], or items are sorted using a scoring function which is a by-product of the learned classifier [8, 9].

Preference learning methods are designed for relational ranks, not categorical ranks. This makes preference learning theoretically more appropriate to be applied to selection problems. They, however, have not been applied to sentence selection task as far as we know and therefore their validity in the task is unknown.

3. Problem Statement

Each past study described in the previous section has its own problem. Hirao et al.'s classifier-based approach showed good performance in real sentence selection experiments, but lacks theoretical justification. Preference learning is theoretically more suitable to selection problems, but has not been applied to sentence selection task and thus its validity for the task is unknown.

To solve these problems, this study will answer the following questions.

- *Is there any theoretical justification for solving selection problem with classification learning? If the answer is yes, can we construct a better learning algorithm based on the theoretical discussion?*
- *What is the best learning methods for sentence selection?*

We will address the first question in Sec. 4 and the second one in Sec. 5.

4. A Large Margin Method for Selection Learning

4.1 Definitions

Selector

Given a set of finite number of items, X, we define *selector* as follows.

1. Selector F is a mapping from the set of non-empty subsets of X, $\{S|S \subset X, S \neq \{\}\}$, to an element of X.
2. $F(S) \in S$.
3. For all $S, T \subset X$, if $T \subset S$ and $F(S) \in T$, then $F(S) = F(T)$

The second condition guarantees that a selector outputs an element of the input set. (This is why we call the function "selector".) Roughly speaking, the third condition says that a selector must choose the same element from a set and its subset. This condition might be easily understood if you interpret that a selector chooses the "best" item from its input and the "best" item must be the same if two item sets share the item and one set is the subset of the other.

Order Induced by Selector F

As the above "best" item interpretation may suggest, a selector F naturally defines an order $\succeq_F$ on X.

1. $x \succ_F y \Leftrightarrow F(\{x, y\}) = x$
2. $x =_F y \Leftrightarrow x = y$
3. $x \succeq_F y \Leftrightarrow x \succ_F y \vee x =_F y$

It is easily proved that $\succeq_F$ satisfies reflexivity, antisymetricity and transitivity, thus is a linear order.

Scoring Function for Selector F

1. Scoring function g for F is a mapping from X to $\mathbb{R}$.
2. $g(x) > g(y) \Leftrightarrow x \succ_F y$

Since $\succeq_F$ is a linear order on X and $|X|$ is finite, there is at least one scoring function for any selector. (Sort X according to $\succeq_F$ and assign decreasing real numbers to the sorted items.)

N-best Selector Induced by Selector F

We (recursively) define *a n-best selector* F_n induced by a selector F as follows. ($\mathcal{X}$ represents the set of all subsets of X.)

1. N-best selector F_n is a a mapping from $\mathcal{X}$ to $\mathcal{X}$.
2. If S is not empty, $F_1(S) = \{F(S)\}$. $F_1(\{\}) = \{\}$.
3. For $n > 1$, $F_n(S) = F_{n-1}(S) \cup F_1(S/F_{n-1}(S))$

It is apparent from the above definition why we call F_n n-best selector; it chooses the "best" items one by one (the last condition) and outputs the n-best items. Please note that $F_n(S) = S$ when $n \geq |S|$. ($|S|$ is the number of elements in S.)

p-percentile Selector Induced by Selector F

Another kind of selector is p-percentile selector, which chooses the items in the top $p \times 100\%$ percentile.

1. p-percentile selector G_p is a a mapping from $\mathcal{X}$ to $\mathcal{X}$. $0 < p \leq 1$.
2. $G_p(S) = F_{\lceil np \rceil}(S)$, where $n = |S|$ and $\lceil a \rceil$ is the smallest integer not less than a.

Please note that the number of the selected items depends on the input size.

4.2 Marginalized Selection Probability

If we can learn a scoring function for the target selector well, it is expected that the selection problem is solved accurately. The learning is not a straight-forward job since we can not observe scoring function's outputs directly. Fortunately, however, one can construct scoring functions if there are enough samples of the selection problem. We will describe three scoring functions, each of which corresponds to each selector in the previous section.

Marginalized Selection Probability of Selector F

The marginalized selection probability $f(x)$ is the probability that the selector F chooses x given an item set S which includes x. [1] That is,

$$f(x) = \Pr\{x = F(S) | x \in S\}. \tag{2.1}$$

As we will prove soon later, the marginalized selection probability is a scoring function of the selector F. Hereafter we assume that input items in S are independently and identically drawn (i.i.d.) from a probability distribution $p(x)$, and its size $|S|$ is independently determined by probability distribution $\Pr\{|S| = m\}$. Additionally it is assumed that $p(x)$ does not concentrate on particular items and $\Pr\{|S| = m\}$ decreases rapidly enough as m gets larger, so that a typical S does not have duplicate members.

Theorem The marginalized selection probability of Eq.(2.1) is a scoring function of the selector F.

[1] Marginalized selection probability is a generalization of order statistic [11] for the order induced by a selector. In our previous work [12], we call the marginalized selection probability "the conditional distribution of generalized order statistic". We have changed the name because we believe that selection-based names and concepts make our discussion much easier to understand.

Proof

$$\begin{aligned} f(x) &= \Pr\{x = F(S) | x \in S\} \\ &= \sum_{m=1}^{\infty} \Pr\{x = F(S) | x \in S, |S| = m\} \Pr\{|S| = m\} \\ &= \sum_{m=1}^{\infty} {}_m\mathrm{C}_1 q(x)^{m-1} \Pr\{|S| = m\} \\ &= \sum_{m=1}^{\infty} a_m q(x)^{m-1}. \end{aligned}$$

Here $a_m = {}_m\mathrm{C}_1 \Pr\{|S| = m\}$ and $q(x) = \sum_{z \in X, x \succ_F z} p(z)$. Since a_m is non-negative and at least one of a_m is strictly positive, $f(x)$ is a monotonically increasing function of $q(x)$. Thus $x \succ_F y \Leftrightarrow q(x) > q(y) \Leftrightarrow f(x) > f(y)$. □

Marginalized Selection Probability of N-best Selector F_n

A similar scoring function can be constructed from a n-best selector. First we define the marginalized selection probability of n-best selector F_n as follows.

$$f_n(x) = \Pr\{x \in F_n(S) | x \in S\} \tag{2.2}$$

Theorem The marginalized selection probability of Eq.(2.2) is a scoring function of the selector F.

Proof

$$\begin{aligned} f_n(x) &= \Pr\{x \in F_n(S) | x \in S\} \\ &= \sum_{m=1}^{\infty} \Pr\{|S| = m\} \Pr\{x \in F_n(S) | x \in S, |S| = m\} \\ &= \sum_{m=1}^{n-1} \Pr\{|S| = m\} \\ &\quad + \sum_{m=n}^{\infty} \Pr\{|S| = m\} \Pr\{x \in F_n(S) | x \in S, |S| = m\} \end{aligned} \tag{2.3}$$

$$
\begin{aligned}
&\Pr\{x \in F_n(S) | x \in S, |S| = m\} \\
&= \Pr\{x \in F_{n-1}(S) \vee x \in F_1(S/F_{n-1}(S)) | x \in S, |S| = m\} \\
&= \Pr\{x \in F_{n-1}(S) | x \in S, |S| = m\} \\
&\quad + \Pr\{x \in F_1(S/F_{n-1}(S)) | x \in S, |S| = m\} \\
&= \sum_{k=1}^{n} \Pr\{x \in F_1(S/F_{k-1}(S)) | x \in S, |S| = m\}, \qquad (2.4)
\end{aligned}
$$

where $F_0(S) \equiv \{\}$. To derive the last equation, we ignore the possibility that S has duplicate members. [2] Each summand in Eq.(2.4) is equal to

$$
\begin{aligned}
&\Pr\{x \in F_1(S/F_{k-1}(S)) | x \in S\} \\
&= \Pr\{x \in F_1(S/F_{k-1}(S)), x \in S/F_{k-1}(S) | x \in S\} \\
&\quad + \Pr\{x \in F_1(S/F_{k-1}(S)), x \notin S/F_{k-1}(S) | x \in S\} \\
&= \Pr\{x = F(S/F_{k-1}(S)), x \in S/F_{k-1}(S) | x \in S\} \\
&= {}_m\mathrm{C}_1 {}_{m-1}\mathrm{C}_{k-1} (1 - q(x))^{k-1} q(x)^{m-k}. \qquad (2.5)
\end{aligned}
$$

The last expression represents the probability that x is the n-th best item.

From Eq.(2.4) and Eq.(2.5), we get

$$
\begin{aligned}
&\Pr\{x \in F_n(S) | x \in S, |S| = m\} \\
&= {}_m\mathrm{C}_1 \sum_{k=1}^{n} {}_{m-1}\mathrm{C}_{k-1} (1 - q(x))^{k-1} q(x)^{m-k}. \qquad (2.6)
\end{aligned}
$$

Eq.(2.6) is a monotonically increasing function of $q(x)$ since the differentiation with respect to $q(x)$ is positive except at a single point.

$$
\begin{aligned}
&\frac{d}{dq} \Pr\{x \in F_n(S) | x \in S, |S| = m\} \\
&= m \sum_{k=1}^{n} {}_{m-1}\mathrm{C}_{k-1} \{-(k-1)(1-q)^{k-2} q^{m-k} + (m-k)(1-q)^{k-1} q^{m-k-1}\} \\
&= m(m-1) \sum_{k=1}^{n} \{-{}_{m-2}\mathrm{C}_{k-2} (1-q)^{k-2} q^{m-k} + {}_{m-2}\mathrm{C}_{k-1} (1-q)^{k-1} q^{m-k-1}\} \\
&= m(m-1) {}_{m-2}\mathrm{C}_{n-1} (1-q)^{n-1} q^{m-n-1} \\
&\geq 0.
\end{aligned}
$$

[2] This is consistent with the assumptions about $p(x)$ and $\Pr\{|S| = n\}$. (See the descriptions below Eq.(2.1).)

Here the last equality holds only when $q(x) = 0$, i.e, x is the "worst" item in X.

From Eq.(2.3) and the monotonicity of Eq.(2.6), $f_n(x)$ is also a monotonic increasing function of $q(x)$. Thus $x \succ_F y \Leftrightarrow q(x) > q(y) \Leftrightarrow f_n(x) > f_n(y)$, i.e, $f_n(x)$ is a scoring function of F. □

Marginalized Selection Probability of p-percentile Selector G_p

The last marginalized selection probability, $g_p(x)$ corresponds to p-percentile selector.

$$g_p(x) = \Pr\{x \in G_p(S) | x \in S\} \tag{2.7}$$

Theorem The marginalized selection probability of Eq.(2.7) is a scoring function of the selector G_p.

Proof

$$\begin{aligned} g_p(x) &= \sum_{m=1}^{\infty} \Pr\{|S| = m\} \Pr\{x \in G_p(S) | x \in S, |S| = m\} \\ &= \sum_{m=1}^{\infty} \Pr\{|S| = m\} \Pr\{x \in F_{\lceil mp \rceil}(S) | x \in S, |S| = m\} \end{aligned}$$

As proved above, $\Pr\{x \in F_{\lceil mp \rceil}(S) | x \in S, |S| = m\}$ is a monotonically increasing function of $q(x)$. Thus $g_p(x)$ is a scoring function. □

Examples of Marginalized Selection Probability

Figure 2.1 shows examples of marginalized selection probability. The probabilities are numerically computed using a selector, one of whose scoring functions is shown in Figure 2.2. We used the uniform distribution over $[-2, 2]$ as the distribution of x and fixed size inputs ($|S| = 5$). It is observed that all the marginalized selection probabilities keep the order of the scoring function although the shapes are skewed.

4.3 Selection as Classification

The previous section proved that the marginalized selection probability $\Pr\{x \in Sel(S) | x \in S\}$ is a scoring function of the selector Sel. This probability can be

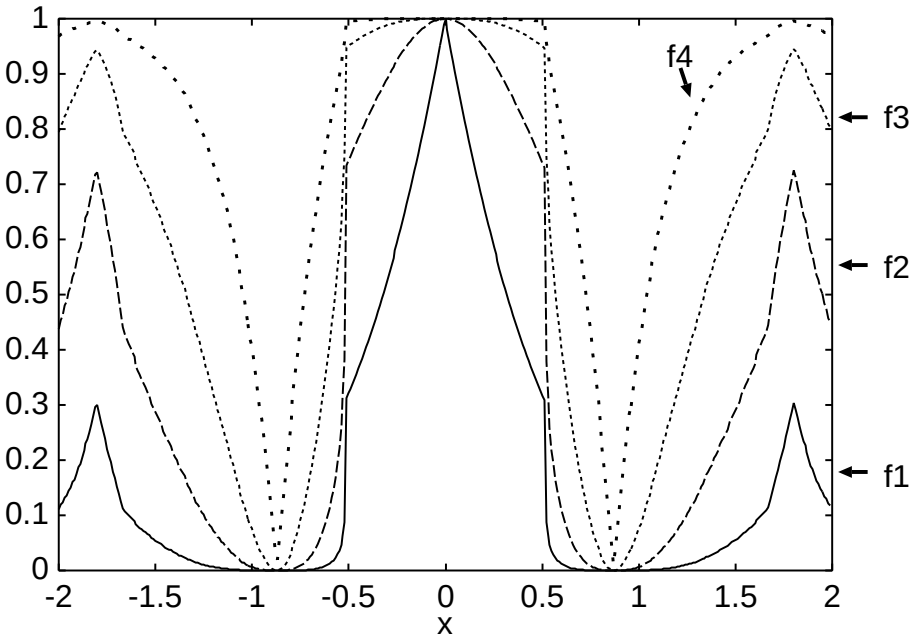


Figure 2.1. Examples of Marginalized Selection Probability. fn is the graph of $f_n(x)$. These probabilities are numerically computed from a selector on $[-2, 2]$, one of whose scoring functions is shown in Figure 2.2. $p(x)$ is the uniform distribution, $\Pr\{|S| = m\} = 1$ for $m = 5$ and 0 otherwise.

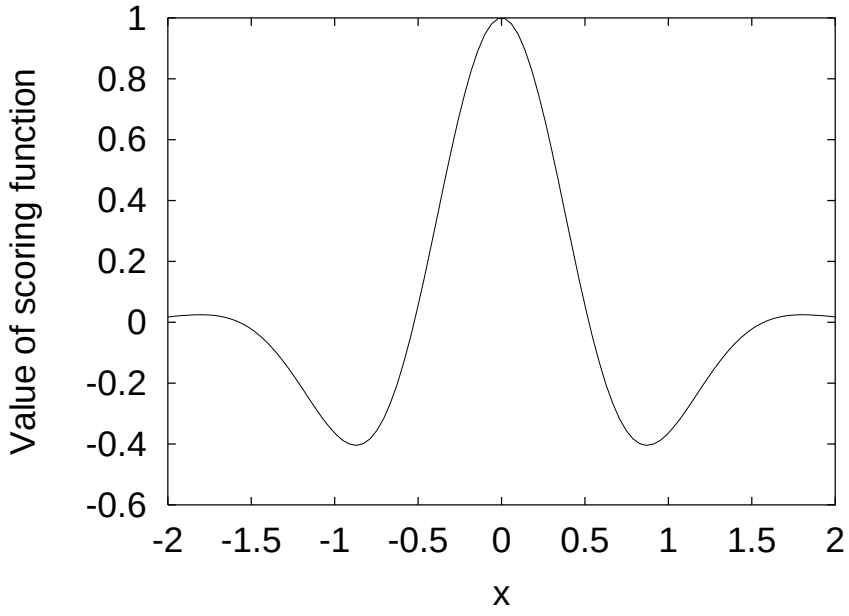


Figure 2.2. A scoring function of the selector used to compute the marginal selection probabilities in Figure 2.1.

also expressed informally as $\Pr\{x \text{ is selected}\}$. This expression shows that we can treat selection like classification in which items are classified into "selected" or "not selected" *as long as our only concern is the marginalized selection probability.* [3] This observation leads to the following reduction from a selection learning problem to a classification learning problem.

Reduction of Selection Learning to Classification Learning

1. Given selection data $D_{\text{sel}} = \{(S_i, Sel(S_i))\}_{i=1}^{m}$, where S_i is a non-empty subset of X, define classification data D_{cls} as

$$\begin{aligned} D_{\text{cls}} \equiv \; & \{(x,y)|x \in X, y \in \{-1,1\}, \\ & \exists i \; x \in S_i \; \wedge y = \mathrm{I}_{\pm}[x \in Sel(S_i)]\}. \end{aligned} \tag{2.8}$$

   Here $\mathrm{I}_{\pm}[statement]$ is an indicator function which is equal to 1 if *statement* is true or -1 otherwise.

2. Estimate the conditional probability $\Pr(y|x)$ from D_{cls}.

3. Output $\Pr(y = 1|x)$ as an estimated scoring function of the selector Sel.

Since the second step in the above reduction is a standard binary classification problem, any classification learning algorithm can be applied to the step as long as its classifier's output can be related to the conditional probability.

Using SVM to Learn a Selector

As an example of the use of classification learner as selection learner, we will explain how Support Vector Machine (SVM) [13] can be used as selection learner.

SVM is known to show good performance in many classification tasks. From a practical point of view, the so-called "kernel trick" is one of the most advantageous characters SVM has since the "trick" enables users to apply SVM to non-vectorial data such as strings and trees. This is especially true for Natural Language Processing in which data is generally non-vectorial. Thus it is expected to be very useful if one can use SVM to learn selector.

[3] It is worth to emphasize here that selection itself is *NOT* a classification: a selector does not choose items based on their membership to a category. If so, the selector could not choose items when its input does not include any element of such category. This would contradict the definition of selector.

Although SVM is usually not regarded as a probabilistic classifier, it is shown in [14] that SVM is (approximately) equivalent to maximum a posteriori (MAP) estimation of the following parametrized class of conditional probabilities.

$$\begin{aligned}
\Pr(y|x;\vec{w},b) &= \frac{\exp(-|1-y(\vec{w}\cdot\vec{\phi}(x)+b)|_+)}{\exp(-|1-(\vec{w}\cdot\vec{\phi}(x)+b)|_+)+\exp(-|1+(\vec{w}\cdot\vec{\phi}(x)+b)|_+)} \\
&\simeq \exp(-|1-y(\vec{w}\cdot\vec{\phi}(x)+b)|_+), \qquad (2.9) \\
\Pr(\vec{w}) &\propto \exp\left(-\frac{1}{2C}\|\vec{w}\|^2\right), \qquad (2.10)
\end{aligned}$$

where $\vec{w} \in \mathbb{R}^N$ and $b \in \mathbb{R}$ are parameters; $\vec{\phi} : X \mapsto \mathbb{R}^N$ is a feature function; $|z|_+$ is defined as z when $z > 0$ and 0 otherwise.

Given samples of $\{(x_i, y_i)\}_{i=1}^m$, the logarithm of a posteriori distribution of the parameters becomes

$$\begin{aligned}
\log \Pr(\vec{w}, b|\{(x_i, y_i)\}_{i=1}^m) &= \log \Pr(\vec{w}) + \sum_{i=1}^m \log Pr(y_i|x_i;\vec{w},b) \\
&\simeq -\frac{1}{2C}\|\vec{w}\|^2 - \sum_{i=1}^m |1-y_i(\vec{w}\cdot\vec{\phi}(x_i)+b)|_+.
\end{aligned}$$

Thus maximum a posteriori value of the parameters is equal to the solution of the following optimization problem.

$$\min_{\vec{w},b} \ \frac{1}{2}\|\vec{w}\|^2 + \sum_{i=1}^m |1-y_i(\vec{w}\cdot\vec{\phi}(x_i)+b)|_+. \qquad (2.11)$$

Eq.(2.11) is equivalent to a standard SVM formalization.

In summary we can get an approximation of a scoring function by training a SVM with the classification data D_{cls} , which is reduced from the selection data, and then computing the conditional probability $\Pr(y|x)$ with Eq.(2.9). If only scoring function is necessary, one can directly use $\vec{w} \cdot \vec{\phi}(x)$ as a scoring function since $\Pr(y = 1|x)$ in Eq.(2.9) is a monotonically increasing function of $\vec{w} \cdot \vec{\phi}(x)$.

We should remind the readers that the idea of solving selection problem by SVM was first proposed by Hirao *et al.* in [5]. They, however, did not provide any theoretical explanation for using classifiers to selection problems. As far as we have known, this work is the first one to give an answer to the theoretical question.

4.4 Selection Support Vector Machine

Successive Selection Problem

In many real situations, selection happens to be repeated. For example, in order to make a ranking of some items, one may select the best item repeatedly. Such situations can be modeled as multiple selectors are applied to the same item set. To represent this more formally, we introduce a new concept "successive selection problem".

> Suppose that $Sel_j (1 \leq j \leq k)$ are n-best selectors F_{n_j} where $n_i < n_j$ for $i < j$, or p-percentile selectors G_{p_j} where $p_i < p_j$ for $i < j$ and all Sel_j are induced from one selector F.
>
> Then *k-successive selection problem* is the estimation problem of the selector F, given samples of $(k+1)$-tuple $(S, Sel_1(S), \ldots, Sel_k(S))$.

One possible solution to a k-successive selection problem is to decompose it to k selection problems, each of which the data $\{(S, Sel_j(S))\}$ is given as the training data. This solution, however, does not count the fact that each Sel_j is induced from the same selector. To overcome this shortcoming, we propose an SVM-based solution which learns successive selection at once.

Selection SVM: Learning Successive Selection at Once

As explained in the previous section, a scoring function of one selector can be learned using an SVM. Thus if we use one SVM for each successive selector Sel_j, we need $2k$ parameters, $\{\vec{w}^j\}_{j=1}^k$ and $\{b^j\}_{j=1}^k$, in total. Since it is assumed that all the successive selector is induced from the same selector, it is natural to expect $\vec{w}^j$ are not so different from each other. This leads to the idea that successive selection problem can be solved more accurately if we learn one SVM for one successive selector but the SVMs' weight vectors $\vec{w}^j$ are constrained to stay near the "center" $\vec{w}$.

Selection SVM

1. Given k-successive selection data,

$$D_{\text{sel}}^k = \{(S_i, Sel_1(S_i), Sel_2(S_i), \ldots, Sel_k(S_k))\}_{i=1}^m,$$

generate the classification data,

$$\begin{aligned} D^k_{\text{cls}} &= \{(x, y^1, y^2, \cdots, y^k) | x \in X, \forall y^j \in \{-1, 1\}, \\ &\qquad \exists\, ix \in S_i \wedge \forall y^j = \mathrm{I}_{\pm}[x \in Sel_j(S_i)]\}. \end{aligned}$$

2. Solve the following optimization problem.

$$\begin{aligned} \min_{\vec{w}, \{\vec{v}^j\}_{j=1}^k, \{b^j\}_{j=1}^k} & \frac{1}{2}\|\vec{w}\|^2 + \sum_{j=1}^k \frac{\lambda}{2}\|\vec{v}^j\|^2 \\ & + C \sum_{(x, y^1, \ldots, y^k) \in D^k_{\text{cls}}} \sum_{j=1}^k |1 - y^j((\vec{w} + \vec{v}^j) \cdot \vec{\phi}(x) + b^j)|_+ \end{aligned} \tag{2.12}$$

3. Output $\vec{w} \cdot \vec{\phi}(x)$ as a scoring function of the selector.

$\vec{v}^j$ represents each weight's deviations from the center weight $\vec{w}$, and is (softly) constrained by the second term in Eq.(2.12). (λ controls the strength of the constraint.)

In practice, the Wolfe dual form of Equation 2.12 is more convenient to solve.

Selection SVM (Dual Form)

1. Generate D^k_{cls} as explained above. We will refer the i-th element of D^k_{cls} as $(x_i, y^1_i, y^2_i, \ldots, y^k_i)(1 \leq i \leq l)$ hereafter.

2. Solve the following optimization problem.

$$\begin{aligned} \max_{\alpha^j_i} \quad & \sum_{i=1}^l \sum_{j=1}^k \alpha^j_i - \frac{1}{2} \sum_{i,i'=1}^l \sum_{j,j'=1}^k \alpha^j_i \alpha^{j'}_{i'} y^j_i y^{j'}_{i'} K(x_i, x_j) \left(1 + \frac{\delta_{j,j'}}{\lambda}\right) \\ \text{s.t.} \quad & 0 \leq \alpha^j_i \leq C \quad \text{for } \forall i, j \\ & \sum_{i=1}^l \alpha^j_i y^j_i = 0 \quad \text{for } \forall j, \end{aligned} \tag{2.13}$$

where $\delta_{t,t'}$ is Kronecker delta and $K(x_i, x_j)(\equiv \vec{\phi}(x_i) \cdot \vec{\phi}(x_j))$ is a kernel function on $X \times X$.

3. Output the following function as a scoring function.

$$\sum_{i=1}^{l}\sum_{j=1}^{k}\alpha_i^j y_i^j K(x_i, x) \tag{2.14}$$

Note that all the computation with x_i can be done using the kernel function $K(x_i, x_j)$ like SVM [13]. Thus we need only a kernel, not an explicit vector representation, on X to use Selection SVM.

5. Experiments

This section reports the results of applying Selection SVM to an artificial data and a sentence selection data. We compared the performance of Selection SVM to SVM [5] and Herbrich's method [9]. We call Herbrich's method "Support Vector Preference Learning (SVPL)" hereafter.

SVPL's inputs are samples of correctly ordered pair, (x, y), in which x is ranked higher (preferred) than y. Given training samples $\{(x_i, y_i)\}_{i=1}^m$ and feature function $\vec{\phi}$, SVPL searches the optimal scoring function $h(x) = \vec{w}\cdot\vec{\phi}(x)$ by which the training sample pairs are separated as much as possible, i.e, $\min\{h(x_i)-h(y_i)\}$ becomes as large as possible. This optimal weight is the solution of the following optimization problem.

$$\min_{\vec{w}} \quad \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{m}|1 - (\vec{w}\cdot\vec{\phi}(x_i) - \vec{w}\cdot\vec{\phi}(y_i))|_+. \tag{2.15}$$

See [9] for more details.

5.1 Artificial Data

Data

We generated the artificial selection data as follows. We let X be $[0,1]\times[0,1]$, and $p(x)$ the uniform distribution on it. The size of S is fixed to 4, i.e, $\Pr\{|S| = m\}$ is equal to 1 if $m = 4$, otherwise 0. The selector F uses the scoring function $(x_1-0.5)(x_2-0.5)$ (x_1 and x_2 is the first and second element of x) and chooses the highest score point from S. Under these setting, we generated samples of successive selection $(S, F_1(S), F_2(S), F_3(S))$, where F_n is the n-best selector induced from F.

Learning of SVM and SVPL

We trained three SVMs, each of which uses $(S, F_j(S))(1 \leq j \leq 3)$ as selection data and learn parameter with the schema of Sec. 4.3.

To train SVPL, the preference data D_{pref} were generated from the successive selection data D^k_{sel}.

$$
\begin{aligned}
D_{\text{pref}} &= D^{\text{sel}}_{\text{pref}} \cup D^{\text{other}}_{\text{pref}} \\
D^{\text{sel}}_{\text{pref}} &= \{(x_1, x_2) | x_1, x_2 \in X, \\
&\quad \exists (S, Sel_1(S), Sel_2(S), \ldots, Sel_k(S)) \in D^k_{\text{sel}}, \\
&\quad\quad \exists i < j, x_1, x_2 \in Sel_j(S) \wedge x_2 \notin Sel_i(S)\} \\
D^{\text{other}}_{\text{pref}} &= \{(x_1, x_2) | x_1, x_2 \in X, \\
&\quad \exists (S, Sel_1(S), Sel_2(S), \ldots, Sel_k(S)) \in D^k_{\text{sel}}, \\
&\quad\quad x_1 \in Sel_k(S) \wedge x_2 \in S \wedge x_2 \notin Sel_k(S)\}
\end{aligned}
$$

We used a RBF kernel $k(x, y) = \exp(-\frac{\|x-y\|^2}{\sigma^2})$ with $\sigma = 0.5$ for Selection SVM, SVM and SVPL. The hyper-parameter C was set to 1 in all the methods, and the hyper-parameter λ of Selection SVM was set to $1, 10^2, 10^4, \infty$. [4]

Performance Measure

The performance was evaluated by n-best accuracy A_n.

$$
A_n = \frac{1}{|D_{\text{test}}|} \sum_{S \in D_{\text{test}}} \frac{|F_n(S) \cap \hat{F}_n(S)|}{|F_n(S)|},
$$

where D_{test} is the test dataset of the successive selection, $\hat{F}_n$ the n-best selector induced from a learned selector. We generated 1,000 test samples of S and used them throughout this experiment.

Results

Figure 2.3 clearly shows the n-best accuracies of Selection SVM, SVMs and SVPL, all of which were trained with 5/10/25/100/250 successive selection samples.[5]

[4] $\lambda = \infty$ means that all $\vec{v}^j$ were fixed to 0 in Equation (2.12).

[5] As for Selection SVM, Figure 2.3 shows only the results for $\lambda = 10^2$. This is because the results with the other λ do not show any significant differences.

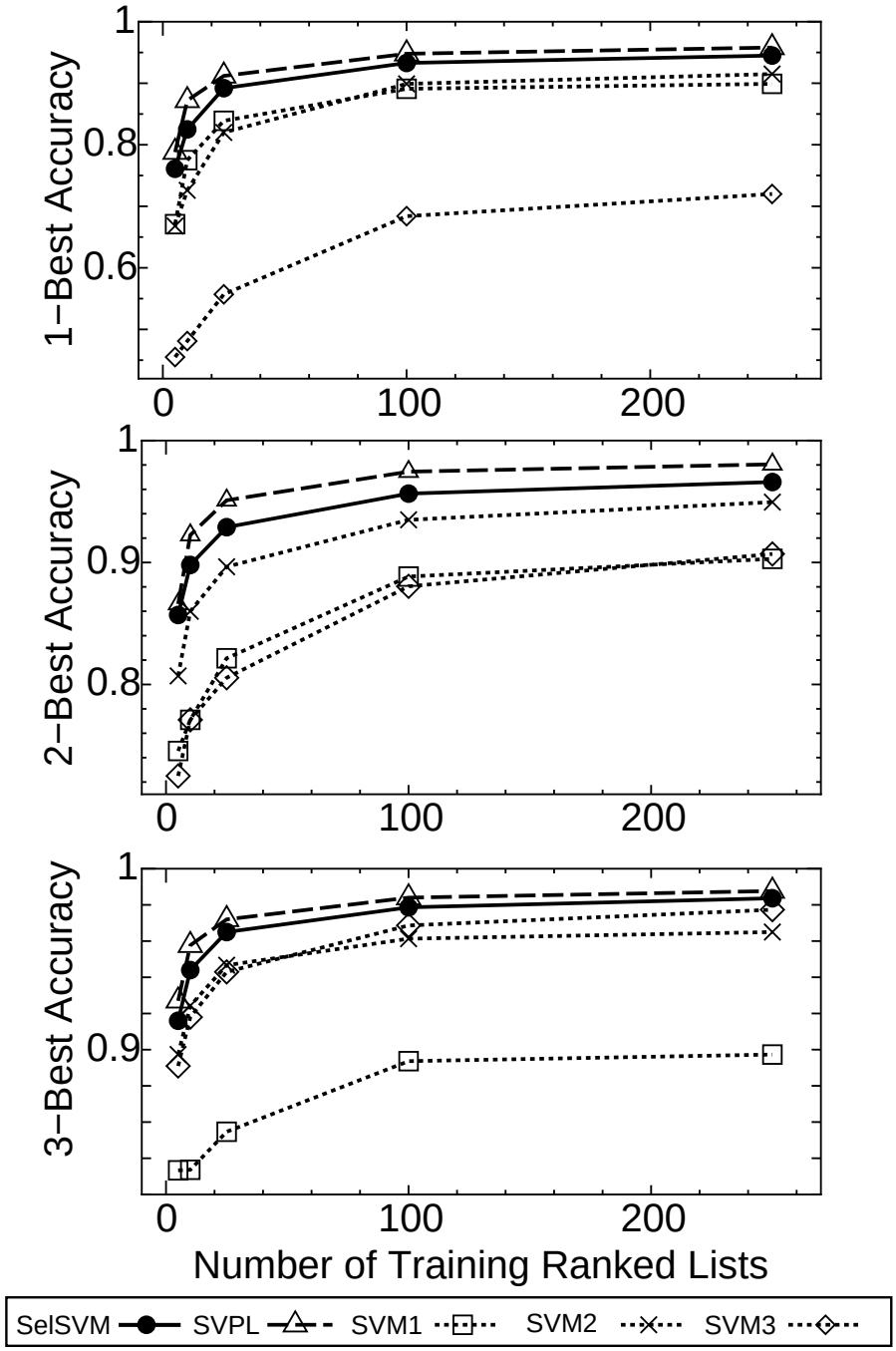


Figure 2.3. The n-best accuracies on artificial data. "SelSVM" is Selection SVM ($\lambda = 10^2$), and "SVMn" SVM estimation of $f_n(x)$.

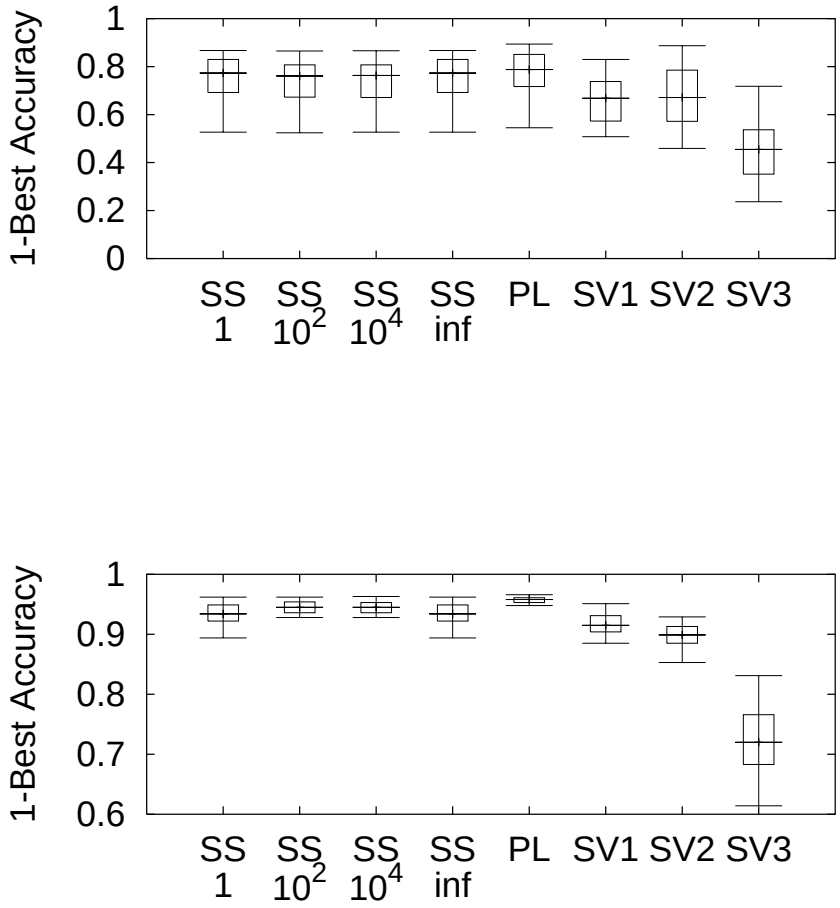


Figure 2.4. The 1-best accuracies of the learners trained on 5 successive selection samples (top) and 250 samples (bottom). SS=Selection SVM, PL=SVPL, SV=SVM. The numbers below "SS" are the values of λ. (inf = ∞)

The training was repeated 100 times and each point in Figure 2.3 is the median of the results. Figure 2.4 shows the box-plots of 1-best accuracies of the learners trained with 5 (top) and 250 (bottom) ranked lists. The box-plots summarize the results of the 100 training results: middle line in the boxes shows the median of 1-best accuracies, the boxes show the ranges from 25% quantile to 75% quantile, and the top and bottom lines show the ranges from 5% quantile to 95% quantile.

Figure 2.3 shows that Selection SVM outperforms SVMs in all n-best accuracies. The differences between those two methods are quite large as shown in Figure 2.4. This is a natural result because Selection SVM uses all the successive selection data for estimation whereas SVMs use only a single selection data. Selection SVM seems to succeed in extracting more information from the same training set.

On the other hand, Selection SVM's accuracies are slightly behind SVPL's accuracies. The differences, however, are not significant compared to the differences between Selection SVM and SVMs, especially for the small training dataset (Figure 2.4).

Selection SVM's accuracies do not show much dependency on the choice of the hyper parameter λ although Figure 2.4 shows that a good choice of λ leads to a smaller variance of the accuracies.

5.2 Real-world Sentence Selection Data

The second experiment was on a real-world sentence selection data, which was used in the First NTCIR's Text Summarization Challenge (TSC-1) [6].

The TSC-1 Data

The TSC-1 dataset is a collection of 180 Japanese newspaper articles, which includes reports, editorials, and commentaries. A small portion (10, 30 and 50%) of sentences of each article were selected by human experts according to the importance of the sentences. Although the experts were not instructed to do so, a larger selection usually contain a smaller selection. Thus the TSC-1 data can be regarded as a successive selection data, where p-percentile selectors ($p = 0.1, 0.3$ and 0.5) are used to choose items (sentences).

[6] NII-NACSIS Test Collection for IR Systems Workshop http://research.nii.ac.jp/ntcir/workshop/

Measure	$A_{0.1}$	$A_{0.3}$	$A_{0.5}$
Selection SVM	$C = 10^{-3}$ $\lambda = 10^{2}$	$C = 10^{-3}$ $\lambda = 10^{2}$	$C = 10^{-3}$ $\lambda = 10^{1}$
SVM(0.1)	$C = 10^{-5}$	$C = 10^{-5}$	$C = 10^{-2}$
SVM(0.3)	$C = 10^{-3}$	$C = 10^{-3}$	$C = 10^{-1}$
SVM(0.5)	$C = 10^{-2}$	$C = 10^{-2}$	$C = 10^{-4}$
SVPL	$C = 10^{-4}$	$C = 10^{-5}$	$C = 10^{-3}$

Table 2.1. The hyper-parameters used in the TSC-1 experiment. SVM(p) is SVM estimation of G_p.

We used the domestic news section of the TSC-1 data, which consists of 63 articles. The 63 articles were shuffled randomly and divided into 43, 10 and 10 articles for training, development and test respectively.

Features

Each sentence was converted into a binary vector in the same fashion as [5]. Elements of the vector represent various aspects of the sentence such as the relative position in article, the number of keywords, the existence of proper nouns, and the type of the functional word at the end of the sentence. The number of elements totaled 318. See [5] for more details of the vector representation.

Training

We used a quadratic kernel $(\vec{x} \cdot \vec{y} + 1)^2$, which was also used in [5], for Selection SVM, SVMs and SVPL. The hyper-parameter C and λ were chosen so that the method shows the best performance on the development data. (We will explain the performance measures later.) We tested $C = 10^n (n = -5, -4, -3, -2, -1, 0)$, and $\lambda = 10^n (n = 0, 1, 2, 3, 4)$ and ∞. Table 2.1 shows the chosen hyper-parameters.

Measure	$A_{0.1}$	$A_{0.3}$	$A_{0.5}$
Selection SVM	0.64	**0.58**	0.64
SVM(0.1%)	**0.68**	**0.58**	0.59
SVM(0.3%)	**0.68**	**0.58**	0.55
SVM(0.5%)	0.45	0.43	**0.65**
SVPL	0.41	0.51	0.53

Table 2.2. p-accuracies in the TSC-data experiment. The best results for each accuracy are emphasized with **bold characters**.

Performance Measures

The performance was evaluated with p-accuracy A_p,

$$A_p = \frac{1}{|D_{\text{test}}|} \sum_{S \in D_{\text{test}}} \frac{|G_p(S) \cap \hat{G}_p(S)|}{|G_p(S)|}.$$

Here $\hat{G}_p$ represents the p-percentile selector induced from a learned scoring function.

Results

Table 2.2 shows p-accuracies for the test articles. The results with TSC-1 dataset are quite different from those with the artificial dataset. Selection SVM and SVMs clearly outperform SVPL although Selection SVM is constantly a good performer over all p-accuracies whereas no single SVM can match this characteristic.

We think that this difference reflects the impact of data sparseness. In the artificial dataset, an item was represented by a two-dimensional vector and a sufficiently large number of items (up to $250 \times 4 = 1000$ items) were fed to the learning algorithms. On the other hand, an item (i.e. a sentence) in TSC-1 dataset was represented by a 318-dimensional vector and only a small number of items (about 900) were fed to the algorithms. With such sparse data, it is important to extract as much information as possible from the training samples.

SVPL only counts preference relations *within* each successive selection whereas Selection SVM and SVMs aggregate instances *across* all selections into "classes" in order to estimate marginalized selection probability. This means that Selection

SVM and SVMs can use extra information SVPL cannot use and may result in producing more accurate scoring functions. We, however, need further study to confirm this conjecture.

6. Conclusion

In this chapter, we first proved that selection problem can be converted into classification problem, thus can be solved by any classifier learning algorithm. Then Selection SVM is proposed to solve successive selection problems. Experiments with an artificial dataset and a sentence selection dataset show that Selection SVM performs well compared to single SVMs and SVPL.

Chapter 3

Multi-topic Text Categorization

1. Introduction

Multi-topic text categorization is a labeling task whose goal is to assign all relevant topics to a text. Topics should be selected from a given topic set and a single text can have multiple topics. There are many examples of multi-topic text categorization in real world applications: news companies use multiple topic tags to classify articles [3]; search engines link multiple categories to a single web page for directory services [4].

Learning of multi-topic text categorization is a learning task in which the learner attempts to estimate the labeling rules from samples of text and assigned topics.

This chapter is organized as follows. In Section 2, we briefly review related works in multi-topic text categorization. In Section 3, we point out a problem of past related works and state this chapter's main problem. Then we propose a new learning algorithm, Maximal Margin Labeling (MML), and also describe efficient algorithms for it in Section 4. MML is tested on datasets of Web pages, and the results are reported in Section 5. Section 6 summarizes this chapter.

2. Related Works

2.1 One Classifier for One Topic

Although there are huge amount of studies on text categorization[1], most studies take the same approach of decomposing a multi-topic categorization problem into multiple binary judgement subproblems in which each text is judged whether it is relevant to a topic.

This decomposition approach is very popular since one can use any binary classifier to judge the relevance. It also shows quite good performances on various text categorization tasks when it is used with an appropriate classifier such as Support Vector Machines [17].

However, this approach ignores the correlation between topics. Thus even when strong correlation exists, it cannot use such correlation to improve its prediction accuracy.

2.2 Parametric Mixture Model

Parametric Mixture Model (PMM) [4] is a probabilistic model of multi-topic text. PMM assumes that a text is a "bag-of-words" and word order is completely ignored[2]. It is also assumed that a multi-topic text is generated by random sampling from a probabilistic distribution of words, and the source distribution of a multi-topic text is a mixture (an average) of topic-specific word distributions. The topic-specific distributions are estimated from sample multi-topic texts.

When predicting label, PMM searches the most likely combination of topics for the bag-of-words of given text. Thus PMM is different from the above decomposition approach in the sense that its prediction is decided on the label basis, not on the topic basis.

PMM's mixture model, however, does not count multi-topic words, which are words strongly relating to multiple topics. Thus when such words often appear in training texts, PMM seems to suffer serious model mismatches.

[1] For example, see the references in [15, 16].

[2] This (rather radical) assumption is widely accepted among text categorization researchers.

2.3 BoosTexter

BoosTexter is a boosting-based algorithm for multi-topic text categorization. [18] Boosting is a general method to get a good classifier by combining simple but not-so-good classifiers (*weak learners*) [19]. In BoosTexter, each weak learner is responsible only for judging a topic's relevance as is the case in the decomposition approaches. On the other hand, the loss function of the combined classifier is label-based, not topic-based. Thus one can say BoosTexter is a hybrid method in which topic-wise classifiers are trained with label-wise evaluation.

When predicting label, BoosTexter judges each topic relevance independently based on the sum of weak learners' scores. Thus the correlation between topics is not taken into account explicitly.

3. Problem Statement

All methods we described in the previous section have a common feature that they somehow decompose a label into topics: binary classification approaches do no care about which label the target topic belongs to; PMM represents multi-topic as a simple aggregation of component topics; BoosTexter uses single-topic classifiers as its building blocks. This label decomposition is quite reasonable from the viewpoints of keeping model simple and making computational costs lower. However, this can be a disadvantage from a viewpoint of performance since there are some cases that important information is lost through label decomposition.

For example, suppose that we have a multi-topic categorization task in which it is known that several topics are mutually exclusive, i.e, they are never assigned together to text. In this case, if we have a strong evidence that one of the exclusive topics is relevant to some texts, we can safely predict that the labels of the texts never include the other exclusive topics. Label decomposition, however, discards information of such strong (negative) correlation.

As another example, suppose that we have a multi-topic categorization problem of scientific papers in which quantum computing papers are assigned multi-topic label "*quantum physics* & *computer science*"[3]. (*quantum physics* and *computer science* are topics in this example.) Then consider the word "qbit". "Qbit" is very specific to quantum computing[4]: it frequently appears in quantum com-

[3]This kind of situation is arguably often the case when things change very rapidly and topic systems cannot catch up the changes.

[4]Qbit is a unit of quantum information.

puting literatures, but rarely seen in other research papers. Thus it is natural to think that clever multi-topic categorization algorithms should use "qbit" as an important evidence to assign label "*quantum physics & computer science*". However, label decomposition makes it difficult to take multi-topic words such as "qbit" into account.

One solution to these problems is to regard *multi-topic* categorization problems as *multi-class* classification problems in which each label is a distinguished class. This means, for example, that if *sports* and *politics* are topics, there are four classes, *sports*, *politics*, *sports & politics*, and a reject class (i.e, no topic is assigned). The above problems with label decomposition are not problems any more for this "label as class" approach: the case of mutually exclusive topics can be naturally represented as a case in which a class which corresponds to a combination of these topics does not have any member; multi-topic words can be represented specific words to the classes of that multi-topics.

This label-as-class approach, however, has its own problems. From n topics, 2^n labels (i.e, classes) can be generated. This implies that training data per class can be very sparse and the data sparseness can easily cause over-fitting. Additionally existing multi-class classifier algorithms simply cannot deal with such a huge number of classes. These problems define the following question we will answer in this chapter.

> *Based on the label-as-class idea, is it possible to learn a multi-class classifier, which is computationally efficient and does not suffer from over-fitting?*

4. Maximal Margin Labeling

4.1 Multi-class Kernel-based Vector Machines

We start from the multi-class classifier proposed in [20]. Hereafter we use the notation given in Table 3.1.

The multi-class classifier in [20] categorizes an object into the class whose prototype vector $\mathbf{m}$ is the closest to the object's feature vector $\mathbf{x}$. By substituting label for class, the classifier can be written as follows.

$$f(\mathbf{x}) = \arg\max_{\lambda \in \Lambda} \langle \mathbf{x}, \mathbf{m}_\lambda \rangle_X \tag{3.1}$$

Symbol	Meaning
$\mathbf{x}(\in \mathbb{R}^d)$	A document vector
$t_1, t_2, \ldots, t_l$	Topics
T	The set of all topics
$L, \lambda(\subset T)$	A label
$L[j]$	The binary representation of L. 1 if $t_j \in L$ and 0 otherwise.
$\Lambda(= 2^T)$	The set of all possible labels
$\{(\mathbf{x}_i, L_i)\}_{i=1}^m$	Training samples

Table 3.1. Notation

where $\langle , \rangle_X$ is the the inner product of $\mathbb{R}^d$, and $\mathbf{m}_\lambda \in \mathbb{R}^d$ is the prototype vector of label λ. The prototype vectors are learned by solving the following maximal margin problem[5].

$$\begin{aligned} &\min_M \frac{1}{2}\|M\|^2 + C \sum_{1 \le i \le m} \sum_{\lambda \in \Lambda, \lambda \neq L_i} \xi_i^\lambda \\ &\text{s.t.} \quad \langle \mathbf{x}_i, \mathbf{m}_{L_i} \rangle_X - \langle \mathbf{x}_i, \mathbf{m}_\lambda \rangle_X \ge 1 - \xi_i^\lambda \quad \text{for} \quad 1 \le i \le m, \forall \lambda \neq L_i, \end{aligned} \tag{3.2}$$

where M is the prototype matrix whose columns are the prototype vectors, and $\|M\|$ is the Frobenius matrix norm of M.

The solution of Eq.(3.2) gives the prototypes which minimize the sum of their squared norms[6] while keeping the smallest margin between the scores of correct and incorrect labels to 1 (with allowing (small) violations ξ_i^λ). This minimization is equivalent to the maximization of the margin while keeping the sum of the prototype vectors' squared norms constant.

Note that Eq. (3.1) and Eq. (3.2) cover not only training samples' labels, but also all possible labels. This is because the labels unseen in training samples may be relevant to test samples. In usual multi-class problems, such unseen labels never exist. In multi-topic text categorization, however, unseen labels often appear in test data. Thus it is necessary to consider all possible labels in Eq. (3.1) and Eq. (3.2) since it is impossible to know which unseen labels are present in the test samples.

[5]In Eq.(3.2), we penalize all violation of the margin constraints. On the other hand, Crammer and Singer penalize only the largest violation of the margin constraint for each training sample [20]. We chose the "penalize-all" approach since it leads to an optimization problem without equality constraints (see Eq.(3.7)), which is much easier to solve than the one in [20].

[6]Note that $\|M\|^2$ is equivalent to the sum of the squared norms of all prototype vectors.

There are two problems with Eq. (3.1) and Eq. (3.2). The first problem is that they involve the prototype vectors of rare or never seen labels. Without the help of prior knowledge about where the prototype vectors should be, it is impossible to obtain appropriate prototype vectors for such labels. The second problem is that these equations are computationally too demanding since they involve combinatorial maximization and summation over all possible labels, whose number can be quite large. (For example, the number is around 2^{30} in the datasets used in our experiments.)

4.2 A Large Margin Algorithm for MTC

In this section, we incorporate some prior knowledge about the location of prototype vectors into Eq. (3.1) and Eq. (3.2), and propose a novel MTC learning algorithm, **Maximal Margin Labeling (MML)**.

As prior knowledge, we simply assume that *the prototype vectors of similar labels should be placed close to each other.* Based on this assumption, we first rewrite Eq. (3.1) to yield

$$f(\mathbf{x}) = \arg\max_{\lambda \in \Lambda} \langle M^T \mathbf{x}, \mathbf{e}_\lambda \rangle_L, \tag{3.3}$$

where $\langle , \rangle_L$ is the inner product of $\mathbb{R}^{|\Lambda|}$ and $\{\mathbf{e}_\lambda\}_{\lambda \in \Lambda}$ is the orthonormal basis of $\mathbb{R}^{|\Lambda|}$. The classifier of Eq. (3.3) can be interpreted as a two-step process: the first step is to map the vector $\mathbf{x}$ into $\mathbb{R}^{|\Lambda|}$ by M^T, and the second step is to find the closest $\mathbf{e}_\lambda$ to image $M^T\mathbf{x}$. Then we replace $\{\mathbf{e}_\lambda\}_{\lambda \in \Lambda}$ with (generally) non-orthogonal vectors $\{\phi(\lambda)\}_{\lambda \in \Lambda}$ whose geometrical configuration reflects label similarity. More formally speaking, we use vectors $\{\phi(\lambda)\}_{\lambda \in \Lambda}$ that satisfy the condition

$$\langle \phi(\lambda_1), \phi(\lambda_2) \rangle_S = S(\lambda_1, \lambda_2) \quad \text{for } \forall \lambda_1, \lambda_2 \in \Lambda, \tag{3.4}$$

where $\langle , \rangle_S$ is an inner product of the vector space spanned by $\{\phi(\lambda)\}_{\lambda \in \Lambda}$, and S is a Mercer kernel [21] on $\Lambda \times \Lambda$ and is a similarity measure between labels. We use V_S to denote the vector space spanned by $\{\phi(\lambda)\}$.

With this replacement, MML's classifier is written as follows.

$$f(\mathbf{x}) = \arg\max_{\lambda \in \Lambda} \langle W\mathbf{x}, \phi(\lambda) \rangle_S, \tag{3.5}$$

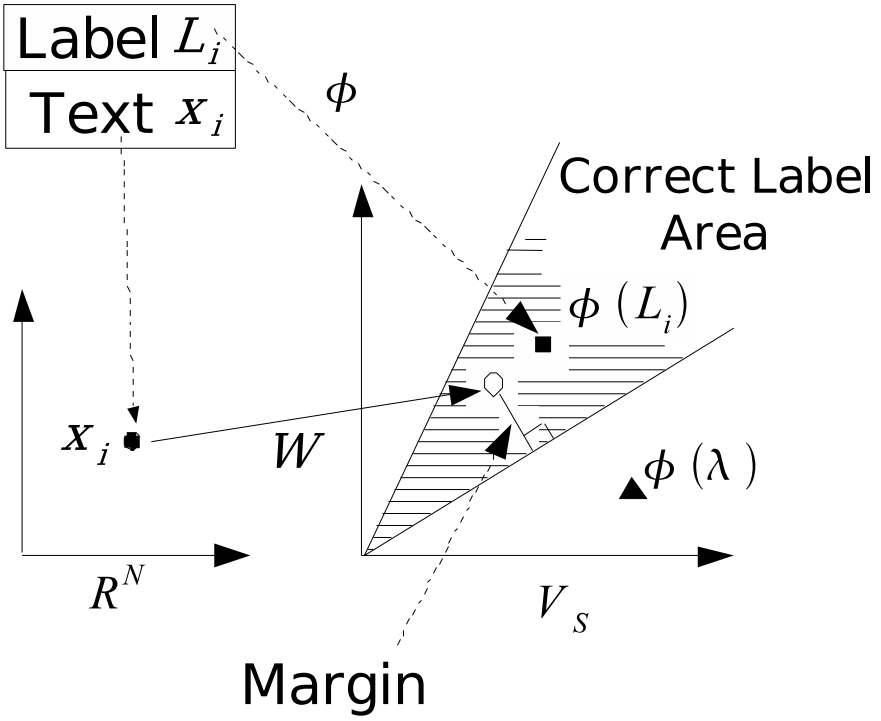


Figure 3.1. Maximal Margin Labeling

where W is a linear map from $\mathbb{R}^d$ to V_S. W is the solution of the following problem.

$$\begin{aligned} \min_{W} \quad & \frac{1}{2}\|W\|^2 + C\sum_{i=1}^{m}\sum_{\lambda\in\Lambda,\lambda\neq L_i}\xi_i^\lambda \\ \text{s.t.} \quad & \left\langle W\mathbf{x}_i, \frac{\phi(L_i)-\phi(\lambda)}{\|\phi(L_i)-\phi(\lambda)\|}\right\rangle \geq 1-\xi_i^\lambda,\ \xi_i^\lambda \geq 0 \\ & \text{for } 1\leq i\leq m, \forall\lambda\neq L_i. \end{aligned} \tag{3.6}$$

Note that if $\phi(\lambda)$ is replaced by $\mathbf{e}_\lambda$, Eq. (3.6) becomes identical to Eq. (3.2) except for a scale factor. Thus Eq. (3.5) and Eq. (3.6) are natural extensions of the multi-class classifier in [20]. We call the MTC classifier of Eq. (3.5) and Eq. (3.6) "Maximal Margin Labeling (MML)".

Figure 3.1 explains the margin (the inner product in Eq. (3.6)) in MML. The margin represents the distance from the image of the training sample $\mathbf{x}_i$ to the boundary between the correct label L_i and wrong label λ. MML optimizes the

linear map W so that the smallest margin between all training samples and all possible labels becomes maximal, along with a penalty C for the case that samples penetrate into the margin.

Dual Form For numerical computation, the following Wolfe dual form of Eq. (3.6) is more convenient.

$$\begin{aligned} \max_{\alpha_i^\lambda} \ & \sum_{i,\lambda} \alpha_i^\lambda - \frac{1}{2} \sum_{i,\lambda} \sum_{i',\lambda'} \alpha_i^\lambda \alpha_{i'}^{\lambda'} (\mathbf{x}_i \cdot \mathbf{x}_{i'}) \frac{S(L_i, L_{i'}) - S(L_i, \lambda') - S(\lambda, L_{i'}) + S(\lambda, \lambda')}{2\sqrt{(1 - S(L_i, \lambda))(1 - S(L_{i'}, \lambda'))}} \\ \text{s.t.} \quad & 0 \leq \alpha_i^\lambda \leq C \quad \text{for } 1 \leq i \leq m, \ \forall \lambda \neq L_i, \end{aligned} \tag{3.7}$$

where we denote $\sum_{i=1}^m \sum_{\lambda \in \Lambda, \forall \lambda \neq L_i}$ by $\sum_{i,\lambda}$, and α_i^λ are the dual variables corresponding to the first inequality constraints in Eq. (3.6). Note that Eq. (3.7) does not contain $\phi(\lambda)$: all the computations involving ϕ can be done through the label similarity S. Additionally $\mathbf{x}_i$ only appears in the inner products, and therefore can be replaced by any kernel of $\mathbf{x}$.

Using the solution α_i^λ of Eq. (3.7), the MML's classifier in Eq. (3.5) can be written as follows.

$$f(\mathbf{x}) = \arg\max_{L \in \Lambda} \sum_{i,\lambda} \alpha_i^\lambda (\mathbf{x} \cdot \mathbf{x}_i) \frac{S(L_i, L) - S(\lambda, L)}{\sqrt{2(1 - S(L_i, \lambda))}}. \tag{3.8}$$

Label Similarity As label similarity, we use Dice measure S_D.

$$S_D(\lambda_1, \lambda_2) = \begin{cases} 1 & \text{if } |\lambda_1| = |\lambda_1| = 0 \\ \frac{2|\lambda_1 \cap \lambda_2|}{|\lambda_1| + |\lambda_2|} & \text{otherwise} \end{cases} \tag{3.9}$$

It can be shown that Dice measure is a Mercer kernel on $\Lambda \times \Lambda$, i.e, it is an inner product of some vectorization of λ. See this chapter's appendix for its proof.

4.3 Approximation in Learning

Eq. (3.7) contains the sum over all possible labels. As the number of topics (l) increases, this summation rapidly becomes intractable since $|\Lambda|$ grows exponentially as 2^l. To circumvent this problem, we approximate the sum over all possible labels in Eq. (3.7) by the partial sum over α_i^λ of $|(A \cap B^c) \cup (A^c \cap B)| = 1$ and set all

the other α_i^λ to zero. This approximation reduces the burden of the summation quite a lot: the number of summands is reduced from 2^l to l, which is a huge reduction especially when many topics exist.

To understand the rationale behind the approximation, first note that α_i^λ is the dual variable corresponding to the first inequality constraint (the margin constraint) in Eq. (3.7). Thus α_i^λ is non-zero if and only if $W\mathbf{x}_i$ falls in the margin between $\phi(L_i)$ and $\phi(\lambda)$. We assume that this margin violation mainly occurs when $\phi(\lambda)$ is "close" to $\phi(L_i)$, i.e. $|(A \cap B^c) \cup (A^c \cap B)| = 1$. If this assumption holds well, the proposed approximation of the sum will lead to a good approximation of the exact solution.

4.4 Polynomial Time Algorithms for Classification

The classification of MML (Eq. (3.8)) involves the combinatorial maximization over all possible labels, so it can be a computationally demanding process. However, efficient classification algorithms are available when the dice measure is used as label similarity.

Eq. (3.8) can be divided into the subproblems by the number of topics in a label.

$$f(\mathbf{x}) = \arg \max_{L \in \{\hat{L}_1, \hat{L}_2, \ldots, \hat{L}_l\}} g(\mathbf{x}, L), \tag{3.10}$$

$$\hat{L}_n = \arg \max_{L \in \Lambda, |L| = n} g(\mathbf{x}, L). \tag{3.11}$$

where $g(\mathbf{x})$ is

$$\begin{aligned}
g(\mathbf{x}, L) &= \begin{cases} a_0 & \text{if } |L| = 0 \\ \sum_{k=1}^{l} a_{|L|}[k]\, L[k] & \text{otherwise,} \end{cases} \\
a_0 &= \sum_{i, \lambda \neq L_i} \frac{\alpha_i^\lambda (\mathbf{x}_i \cdot \mathbf{x})}{\sqrt{2(1 - D(L_i, \lambda))}} \\
&\qquad \times 2(I[|L_i| = 0] - I[|\lambda| = 0]), \\
a_n[k] &= \sum_{i, \lambda \neq L_i} \frac{\alpha_i^\lambda (\mathbf{x}_i \cdot \mathbf{x})}{\sqrt{2(1 - D(L_i, \lambda))}} \\
&\qquad \times \left(\frac{2L_i[k]}{|L_i| + n} - \frac{2\lambda[k]}{|\lambda| + n} \right).
\end{aligned} \tag{3.12}$$

Here $n = |L|$. The computational cost of Eq. (3.12) for all j is $O(n_\alpha l)$ (n_α is the number of non-zero α), and that of Eq. (3.11) is $O(l \log l)$. Thus the total cost

of the classification by Eq. (3.10) is $O(n_\alpha l^2 + l^2 \log l)$. On the other hand, n_α is $O(ml)$ under the approximation described above. Therefore, the classification can be done within $O(ml^3)$ computational steps, which is a significant reduction from the case that the brute force search is used in Eq. (3.8).

5. Experiments

In this section, we report experiments that compare MML to SVM, k nearest neighbor (kNN), PMM and BoosTexter using a collection of Web pages [4]. We chose these methods to compare since (1) SVM and kNN are two of the most popular binary classifiers in text categorization studies, and (2) PMM and BoosTexter take different approaches from these binary classifiers.

5.1 Web Datasets

The Web datasets used in our experiment are the same as those used in [4]. The datasets consist of 11 sets of Web pages which were collected through the hyperlinks from Yahoo!'s directory service (`dir.yahoo.com`). Each dataset corresponds to one of the top categories listed in the directory services. Pages in the datasets are labeled with Yahoo's *second* level sub-categories from which the pages are hyperlinked. Thus, theses sub-categories are *topics* in our term. Table 3.2 shows a summary of the dataset.

From each dataset, we randomly selected 500 pages for development data and 3,000 pages for test data The rest of the pages were pooled for training data of various sizes.

5.2 Feature Vector

As feature vector, we used "bag-of-words" vector, each of whose element represents the "weight" of a word in the text. Since the purpose of our experiments is to compare the learning algorithms, we neither performed any stemming nor removed any stop-words. We tested three types of weighting.

Binary If the word is present in the text, the weight is 1. Otherwise 0.

TF The weight is equal to the number of appearances of the term in the text (term frequency).

Dataset Name (Abbrev.)	#Text	#Voc	#Tpc	#Lbl	Label Size Frequency (%)				
					1	2	3	4	≥5
Arts & Humanities (Ar)	7,484	23,146	26	599	55.6	30.5	9.7	2.8	1.4
Business & Economy (Bu)	11,214	21,924	30	233	57.6	28.8	11.0	1.7	0.8
Computers & Internet (Co)	12,444	34,096	33	428	69.8	18.2	7.8	3.0	1.1
Education (Ed)	12,030	27,534	33	511	66.9	23.4	7.3	1.9	0.6
Entertainment (En)	12,730	32,001	21	337	72.3	21.1	4.5	1.0	1.1
Health (He)	9,205	30,605	32	335	53.2	34.0	9.5	2.4	0.9
Recreation (Rc)	12,828	30,324	22	530	69.2	23.1	5.6	1.4	0.6
Reference (Rf)	8,027	39,679	33	275	85.5	12.6	1.5	0.3	0.1
Science (Si)	6,428	37,187	40	457	68.0	22.3	7.3	1.9	0.5
Social Science (SS)	12,111	52,350	39	361	78.4	17.0	3.7	0.7	0.3
Society & Culture (SC)	14,512	31,802	27	1054	59.6	26.1	9.2	2.9	2.2

Table 3.2. A summary of the web page datasets. "#Text" is the number of texts in the dataset, "#Voc" the number of vocabularies (i.e. features), "#Tpc" the number of topics, "#Lbl" the number of labels, and "Label Size Frequency" is the relative frequency of each label size in the dataset. (Label size is the number of topics in a label.)

TF×IDF The weight is equal to the product of TF and inverse document frequency (IDF) [22]. IDF is $\log(N_d/N_w)$, where N_d is the number of documents in the dataset and N_w the number of documents in which the word appears. IDF gives high weight to rarely seen words.

5.3 Learning Setup

MML

We used a normalized linear kernel $k(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'/\|\mathbf{x}\|\|\mathbf{x}'\|$.

SVM

For each topic, an SVM classifier is trained to predict whether the topic is relevant (positive) or irrelevant (negative) to input documents. Different penalty parameters were used for positive and negative examples as proposed in [23]: the ratio of the penalties for positive and negative examples was set equal to the ratio of the numbers of negative and positive examples.

We used a normalized linear kernel $k(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'/\|\mathbf{x}\|\|\mathbf{x}'\|$.

Method	Feature Type	Parameter
MML	TF, TF×IDF	C = 0.1, 1, 10
PMM	TF	Model1, Model2
SVM	TF, TF×IDF	C = 0.1, 1, 10
Boost	Binary	$R = \{2, 4, 6, 8, 10\} \times 10^3$
kNN	TF, TF×IDF	k = 1, 3, 5, 7

Table 3.3. Candidate feature types and learning parameters. (R is the number of weak hypotheses.) The underlined fetures and parameters were selected for the evaluation with the test data.

k nearest neighbor

We normalized feature vectors before computing distance between inputs, which is equivalent to the usage of normalized linear kernel in MML and SVM.

BoosTexter

BoosTexter has four variants[18]. We used "real abstaining AdaBoost.MH", which showed the best performance in the experiments of [18].

5.4 Feature and Parameter Selection

Before testing, we searched the best combinations of feature types and learning parameters as follows.

1. Train the learners on 2,000 pages randomly drawn from the pooled dataset with all combinations of feature types and parameters listed in Table 3.3.
2. Compute average Dice measure (Sec.5.5) of each feature-parameter combination using the development dataset.
3. Choose the combination of highest average Dice measure.

The combinations underlined in Table 3.3 were chosen by the above procedure.

5.5 Evaluation Measures

We used three measures to evaluate labeling performance: micro average F-measure, macro average F-measure and average Dice measure. In the follow-

ing definitions, $\{L_i^{\text{pred}}\}_{i=1}^n$ and $\{L_i^{\text{true}}\}_{i=1}^n$ mean the predicted labels and the true labels, respectively.

Micro Average F-measure $\bar{F}_{\text{micro}}$

$$\bar{F}_{\text{micro}} = \frac{2\sum_{i=1}^n |L_i^{\text{true}} \cap L_i^{\text{pred}}|}{\sum_{i=1}^n |L_i^{\text{true}}| + \sum_{i=1}^n |L_i^{\text{pred}}|} \tag{3.13}$$

Micro average F-measure is a standard evaluation measure in text categorization and information retrieval citeIR-textbook. $\bar{F}_{\text{micro}}$ evaluates the labeling accuracy regarding the whole dataset as single item.

Macro Average F-measure $\bar{F}_{\text{macro}}$

$$\bar{F}_{\text{macro}} = \frac{1}{l}\sum_{j=1}^{l} \frac{2\sum_{i=1}^n L_i^{\text{pred}}[j] L_i^{\text{true}}[j]}{\sum_{i=1}^n (L_i^{\text{pred}}[j] + L_i^{\text{true}}[j])}. \tag{3.14}$$

Macro average F-measure is also a standard evaluation measure in text categorization and information retrieval citeIR-textbook. $\bar{F}_{\text{macro}}$ evaluates the average accuracy of labeling for each *topic*.

Average Dice Measure $\bar{D}$

$$\bar{D} = \frac{1}{n}\sum_{i=1}^{n} \frac{2|L_i^{\text{pred}} \cap L_i^{\text{true}}|}{|L_i^{\text{pred}}| + |L_i^{\text{true}}|} \tag{3.15}$$

This measure was used in [4]. $\bar{D}$ evaluates the average accuracy of labeling for each *item*.

5.6 Results

Performance Comparison

First we trained the classifiers with 2,000 samples randomly chosen from the pooled data. We then calculated the three evaluation measures on the test data. This process was repeated five times and the measures were averaged. Table 5.6 shows the results. Table 5.6 clearly shows that MML outperforms other methods

	$\bar{F}_{\text{micro}}$					$\bar{F}_{\text{macro}}$					$\bar{D}$				
	MM	SV	PM	kN	Bo	MM	SV	PM	kN	Bo	MM	SV	PM	kN	Bo
Ar	**0.52**	0.47	0.44	0.39	0.39	**0.30**	0.29	0.24	0.26	0.22	**0.55**	0.46	0.50	0.42	0.38
Bu	**0.76**	0.75	0.70	0.70	0.71	0.25	0.29	0.20	**0.33**	0.20	**0.80**	0.76	0.75	0.74	0.75
Co	**0.59**	0.58	0.54	0.50	0.48	0.27	0.30	0.19	**0.31**	0.17	**0.62**	0.55	0.61	0.53	0.47
Ed	**0.55**	0.52	0.48	0.44	0.42	0.25	**0.25**	0.21	0.25	0.16	**0.56**	0.48	0.51	0.46	0.37
En	**0.62**	0.58	0.54	0.50	0.51	**0.37**	0.35	0.30	0.35	0.29	**0.64**	0.54	0.61	0.52	0.49
He	**0.71**	0.68	0.61	0.57	0.60	0.35	**0.35**	0.23	0.32	0.26	**0.74**	0.67	0.66	0.60	0.60
Rc	**0.60**	0.52	0.49	0.47	0.47	**0.47**	0.40	0.36	0.40	0.33	**0.63**	0.49	0.55	0.51	0.44
Rf	**0.65**	0.62	0.56	0.54	0.53	0.29	0.25	0.24	**0.29**	0.16	**0.67**	0.56	0.63	0.56	0.50
Si	**0.59**	0.52	0.46	0.46	0.42	**0.37**	0.31	0.28	0.32	0.19	**0.61**	0.47	0.52	0.49	0.39
SS	**0.69**	0.67	0.58	0.56	0.58	**0.36**	0.31	0.18	0.32	0.15	**0.73**	0.64	0.66	0.60	0.59
SC	**0.55**	0.49	0.48	0.43	0.43	**0.29**	0.26	0.25	0.27	0.20	**0.60**	0.49	0.54	0.47	0.44
Avg	**0.62**	0.58	0.53	0.51	0.50	**0.32**	0.31	0.24	0.31	0.21	**0.65**	0.56	0.59	0.54	0.49

Table 3.4. The micro-average F-measure $\bar{F}_{\text{micro}}$, the macro-average F-measure $\bar{F}_{\text{macro}}$, and the average Dice measure $\bar{D}$. **The bold numbers** (the underlined numbers) are the best ones (the second best ones) of each measure. (MM=MML, SV=SVM, PM=PMM, kN=k-nearest neighbor, Bo=BoosTexter)

	MML	SVM	PMM	kNN	Boost
Ar	**0.44**	0.29	0.21	0.27	0.22
Bu	**0.63**	0.57	0.48	0.52	0.53
Co	**0.51**	0.41	0.35	0.39	0.34
Ed	**0.45**	0.30	0.19	0.32	0.23
En	**0.55**	0.42	0.31	0.40	0.36
He	**0.58**	0.47	0.34	0.40	0.39
Rc	**0.54**	0.37	0.25	0.39	0.33
Rf	**0.60**	0.49	0.39	0.46	0.41
Si	**0.52**	0.36	0.22	0.38	0.28
SS	**0.65**	0.55	0.45	0.48	0.49
SC	**0.44**	0.32	0.21	0.27	0.27
Avg	**0.54**	0.41	0.31	0.39	0.35
Diff. from $\bar{D}$	**0.08**	0.15	0.28	0.15	0.14

Table 3.5. The exact match ratios.

in micro average F-measure and average Dice measure. MML also shows the best performance in macro average F-measure although the margins to the other methods are not as large as observed in macro average F-measure and average Dice measure. Note that all the methods other than MML show good performance in some measures, but also show poor performance in other measures. For example, PMM shows high average Dice measures, but its performance is rather poor when evaluated in macro average F measure.

Table 3.5 shows the ratio of the cases when the predicted labels are exactly the same as the true labels. It is observed that MML predicts exact label more often than other methods. This exactness seems to contribute a lot to MML's accurate labeling since the difference between exact match ratio and average Dice measure, which can be seen as the sum of the exact match ratio and the accuracies on partial matches, is very small.

To see generalization power, we evaluated the classifiers trained with 250–2000 training samples. Figure 3.2 shows each measure averaged over all datasets. It is observed that the MMLs show high generalization even when training data is small.

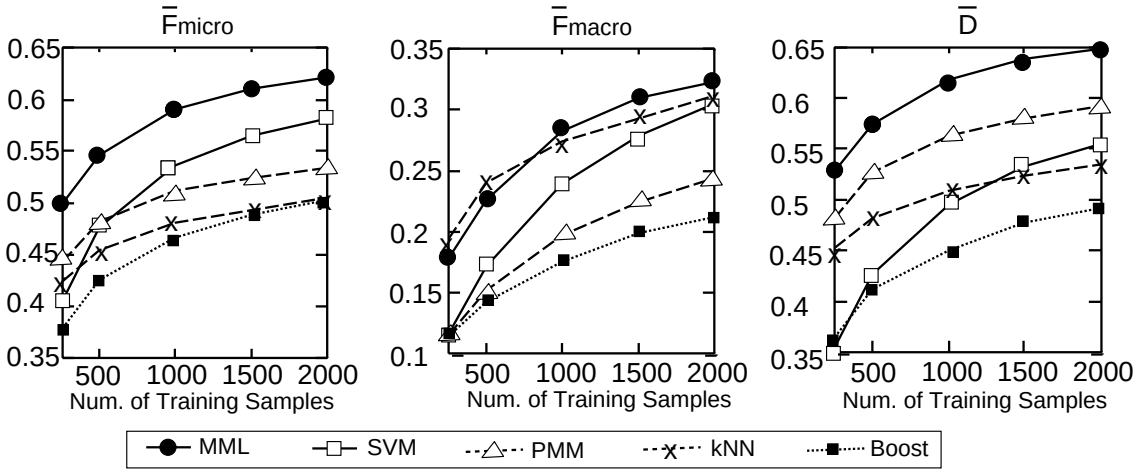


Figure 3.2. The learning curve of micro average F-measure ($\bar{F}_{\text{micro}}$), macro average F-measure ($\bar{F}_{\text{macro}}$), and average Dice measure ($\bar{Dice}$)

Effects of Approximation

As explained in Sec. 4.3, MML uses an approximation in learning to reduce training time. Although this approximation is essential when the number of topics is large, it is possible to train MML without the approximation when the number of topics is small.

Table 3.6 shows the micro average F-measures of MML with approximation (MML) and MML without approximation (MML-all), both of which were trained on 2,000 randomly sampled data in which only 10 most frequent topics are assigned. (Table 3.7) Please note that some texts are not assigned any topics since less frequent topics are removed. Table 3.6 also shows the ratio of the texts that MML and MML-all predict the same label. We also recorded the training times of MML and MML-all, which ran on a Linux box with Intel Xeon 3.4GHz CPU. These times are shown in Table 3.8.

It is observed that there is no major difference between micro average F-measures of MML and MML-all, and their predictions are very similar. On the other hand, training time is greatly reduced by the approximation. This implies that the approximation in Sec.4.3 leads to a similar solution to the original solution while reducing training runtime drastically.

	MML	MML-all	Same Label (%)
Ar	0.55	0.54	78.7%
Bu	0.80	0.80	92.3%
Co	0.64	0.65	86.4%
Ed	0.57	0.56	79.7%
En	0.64	0.64	85.3%
He	0.73	0.72	88.0%
Rc	0.62	0.62	85.0%
Rf	0.69	0.70	89.1%
Si	0.61	0.60	86.7%
SS	0.73	0.73	91.1%
SC	0.60	0.59	86.4%
Avg	0.65	0.65	86.2%

Table 3.6. A comparison of the micro-average F-measures of approximate learning and exact learning.

	#Label	Label Size Freq. (%)				
		0	1	2	3	$\geq$4
Ar	152	7.9	61.3	23.7	5.4	1.7
Bu	54	2.5	61.4	26.7	8.2	1.2
Co	75	7.4	68.9	19.0	4.0	0.7
Ed	129	6.4	62.5	23.2	6.2	1.8
En	105	2.2	74.9	18.2	3.4	1.3
He	99	2.1	57.6	31.4	6.7	2.3
Rc	118	9.7	71.1	15.3	3.1	0.7
Rf	68	9.4	81.1	8.8	0.5	0.1
Si	91	17.6	65.6	13.9	2.3	0.6
SS	79	5.9	79.4	12.6	1.6	0.4
SC	168	8.7	63.4	19.9	5.4	2.6

Table 3.7. A summary of the Web page datasets with the most frequent 10 topics.

	Training Time (sec)		
	MML(T)	MML-all(T_a)	T_a/T
Ar	361	42857	119
Bu	246	21796	87
Co	366	35123	96
Ed	352	46481	132
En	398	39666	99
He	295	32506	110
Rc	399	40372	101
Rf	390	34007	87
Si	434	33811	78
SS	354	30451	86
SC	456	45388	100

Table 3.8. A comparison of the training time of approximate learning (MML) and exact learning (MML-all).

6. Conclusion

In this chapter, we proposed a novel learning algorithm for multi-topic text categorization. The algorithm, Maximal Margin Labeling, embeds labels (sets of topics) into a similarity-induced vector space, and learns a large margin classifier in the space. To overcome the demanding computational cost of MML, we provide an approximation method in learning and efficient classification algorithms. In experiments on a collection of Web pages, MML outperformed other methods including SVM and showed better generalization.

Appendix

Proof that Dice Measure is a Kernel

To prove that S_D is a Mercer kernel on $\Lambda \times \Lambda$, it is sufficient to show that the matrix G ($G_{i+1,j+1} = S_D(\lambda_i, \lambda_j)$) is positive semi-definite. (cf. Proposition 3.5 in [24]). Here $0 \leq i, j \leq 2^l - 1$ and $\{\lambda_i\}_{i=0}^{2^l-1}$ is an enumeration of all labels in Λ.

Without loss of generality, we assume λ_0 is empty. Then G is a matrix of the

following form.

$$G = \begin{pmatrix} 1 & O_{1,2^l-1} \\ O_{2^l-1,1} & H \end{pmatrix}.$$

Here $O_{m,n}$ is $m \times n$ zero matrix and $H_{ij} = S_D(\lambda_i, \lambda_j)$. Since $\mathbf{u}^T G \mathbf{u} = v^2 + \mathbf{w}^T H \mathbf{w}$ for all $\mathbf{u} = (v, \mathbf{w}) \in \mathbb{R}^{2^l}$, G is positive semi-definite if and only if H is positive semi-definite.

H_{ij} is equal to the following P_{ij} and Q_{ij}.

$$\begin{aligned} H_{ij} &= 2P_{ij}Q_{ij} \\ P_{ij} &= |\lambda_i \cap \lambda_j| \\ Q_{ij} &= \frac{1}{|\lambda_i| + |\lambda_j|} \end{aligned}$$

From Proposition 13.2 in [25], if the matrices P and Q are positive semi-definite, H is also positive semi-definite.

First we will prove that P is positive semi-definite. P can be decomposed into a product of vectors.

$$\begin{aligned} P_{ij} &= \mathbf{p}_i^T \mathbf{p}_j \\ \mathbf{p}_i &= (\lambda_i[1], \lambda_i[2], \ldots, \lambda_i[l])^T \end{aligned}$$

Then P is positive semi-definite since

$$\mathbf{w}^T P \mathbf{w} = \sum_{ij} w_i w_j \mathbf{p}_i^T \mathbf{p}_j = \left\| \sum_i w_i \mathbf{p}_i \right\|^2 \geq 0.$$

To prove Q is positive semi-definite, we use the following lemma.

Lemma For a symmetric matrix A which has two identical rows (this also implies A has two identical columns), we denote by $\tilde{A}$ another symmetric matrix which is created from A by removing one of the identical rows and columns. Then A is positive semi-definite if and only if $\tilde{A}$ is positive semi-definite.

Proof of Lemma Without loss of generality, we assume that the last two rows and columns of A are identical. Then A and $\tilde{A}$ are connected by a $(n-1) \times n$

matrix S as follows.

$$
\begin{aligned}
A &= S^T \tilde{A} S, \\
S &= \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 \\ 0 & \cdots & 0 & 1 & 1 \end{pmatrix}
\end{aligned}
$$

From this equation and the fact that for any $\mathbf{y} \in \mathbb{R}^{n-1}$, there exists $\mathbf{x} \in \mathbb{R}^n$ which satisfies $\mathbf{y} = \mathbf{S}\mathbf{x}$,

$$
\begin{aligned}
A \text{ is positive semi-definite} &\Leftrightarrow \mathbf{x} A \mathbf{x}^T \geq 0 \quad \text{for } \forall \mathbf{x} \in \mathbb{R}^n \\
&\Leftrightarrow \mathbf{y} \tilde{A} \mathbf{y}^T \geq 0 \quad \text{for } \forall \mathbf{y} \in \mathbb{R}^{n-1} \\
&\Leftrightarrow \tilde{A} \text{ is positive semi-definite.}
\end{aligned}
$$

This completes the proof of the lemma. □

From the above lemma and the fact that interchanging i-th and j-th rows, and i-th and j-th columns of a matrix does not change positive semi-definiteness of the matrix, Q is positive semi-definite if and only if the following R is positive semi-definite.

$$
R = \begin{pmatrix} \frac{1}{1+1} & \frac{1}{1+2} & \cdots & \frac{1}{1+l} \\ \frac{1}{2+1} & \frac{1}{2+2} & \cdots & \frac{1}{2+l} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{l+1} & \frac{1}{l+2} & \cdots & \frac{1}{l+l} \end{pmatrix}.
$$

As shown in page 137 in [26], R is positive semi-definite if all leading principal minors $|R^{1\cdots k}_{1\cdots k}|$ are positive.

$$
\left|R^{1\cdots k}_{1\cdots k}\right| = \begin{vmatrix} \frac{1}{1+1} & \frac{1}{1+2} & \cdots & \frac{1}{1+k} \\ \frac{1}{2+1} & \frac{1}{2+2} & \cdots & \frac{1}{2+k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{k+1} & \frac{1}{k+2} & \cdots & \frac{1}{k+k} \end{vmatrix}. \tag{3.16}
$$

From page 87 in [26], Eq.(3.16) is equal to

$$
\left|R^{1\cdots k}_{1\cdots k}\right| = \frac{\left\{\prod_{1\leq i<j\leq k}(i-j)\right\}^2}{\prod_{1\leq i,j\leq k}(i+j)} > 0.
$$

Thus R is positive semi-definite. This completes the proof of the claim that S_D is positive semi-definite. □

Chapter 4

Sequence Tagging

1. Introduction

The goal of sequence tagging is to choose a relevant tag from a given tag set for each word in a sequence. Many important NLP problems such as token segmentation, part-of-speech tagging and shallow parsing can be regarded as sequence tagging problems.

The goal of sequence tagging learning is to estimate tagging rules from samples of tagged sequences. In some tasks, samples are converted into a form of tagged sequence before learning. For example, in token segmentation task, whose goal is to segment a sequence into tokens, a segmented sentence is converted into a character sequence in which each character has a tag indicating whether the character is at the beginning or the end of a token. Using this kind of conversion, many NLP tasks can be solved as a sequence tagging task.

This chapter is organized as follows. In Section 2, related works are briefly reviewed. Then we state the main problem of this chapter **tagging strategy learning**, in Section 3. In Section 4, we present an efficient algorithm for computing an optimal tagging order and show an upper bound of tagging accuracies. In Section 5, we proposed a learning algorithm of strategy function, which is a predictor of the best tagging position. The experiments using a real sequence tagging data are reported in Section 6. Then we summarize this chapter in Section 7.

2. Related Works

There are many research papers on sequence tagging. Among them we briefly review two types of tagging models: sequential classification models and one-shot prediction models. These models are chosen here because (1) they are flexible enough to be applied to various sequence tagging problems and (2) they show quite good performance on a wide range of tagging tasks in NLP.

2.1 Sequential Classification Models

Sequential classification models [27, 28, 29, 30, 31] decomposes a tagging task into sequential subtasks. In each subtask, a classifier assigns a tag to a word using the previous assigned tags, in addition to the feature of the word itself, as features.

For example, a typical sequential classification model, "left-to-right" model, first determines the tag of the left-most place and then determines the other tags from left to right using the left tag as an additional feature (Figure 4.1). Similarly "right-to-left" model first determines the right-most tag and then go from right to left using the right tag as an additional feature. [1]

Tsuruoka *et al.* proposed the easiest-first model in which classification order is dynamically computed.[31] The easiest-first model proceeds as follows (Figure 4.2). (a) All tags are temporarily predicted using only word features. (b) The most "confident" (i.e, easiest to predict) tentative tag is actually assigned. (c) The tentative tags adjacent to the assigned one is re-predicted using word feature and the assigned tag as features, and the most confident tentative tag is assigned. (d) This process is repeated until all tags are assigned. Note that tags may be predicted using the both sides tag as features during easiest-first process.

Sequential classification models have two advantages over one-shot prediction models [31]. The first advantage is that they generally run faster than one-shot prediction models. This is because sequential classification models use only actually assigned tags as a part of features and do not need to scan all possible tag assignments, which is the case in one-shot prediction models as we will describe later. The second advantage is that sequential classification models can use any type of classifier as tag predictors. This allows users to improve tagging performance easily by incorporating state-of-the-art machine learning techniques into the models.

[1] In [29], left-to-right model is called "forward parsing", and right-to-left "backward parsing".

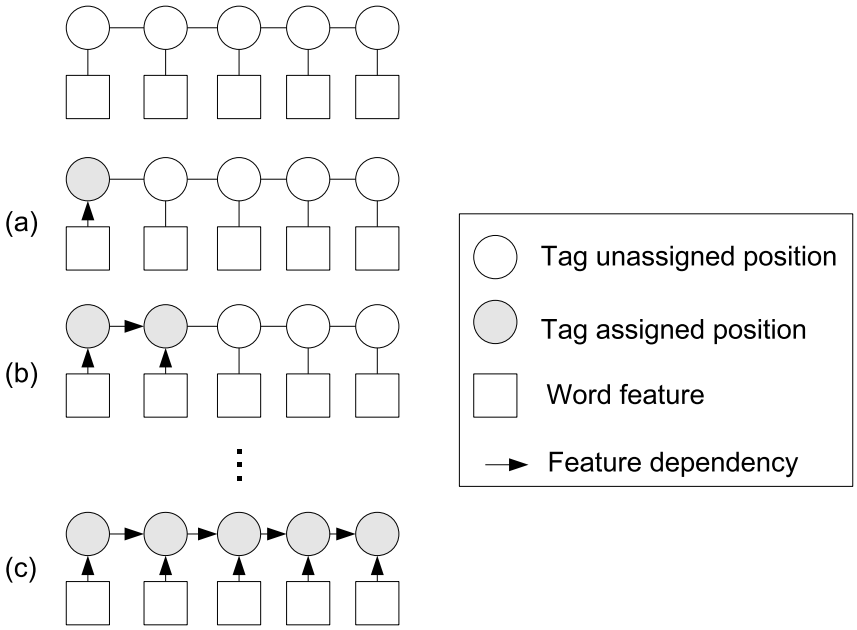


Figure 4.1. Left-to-right sequential model. (a) Assigns the left-most tag using the left-most word feature. (b) Assigns the second left-most tag using the corresponding word feature and the left tag, which is assigned in the previous step. (c) Repeat the assignment step until the right-most tag.

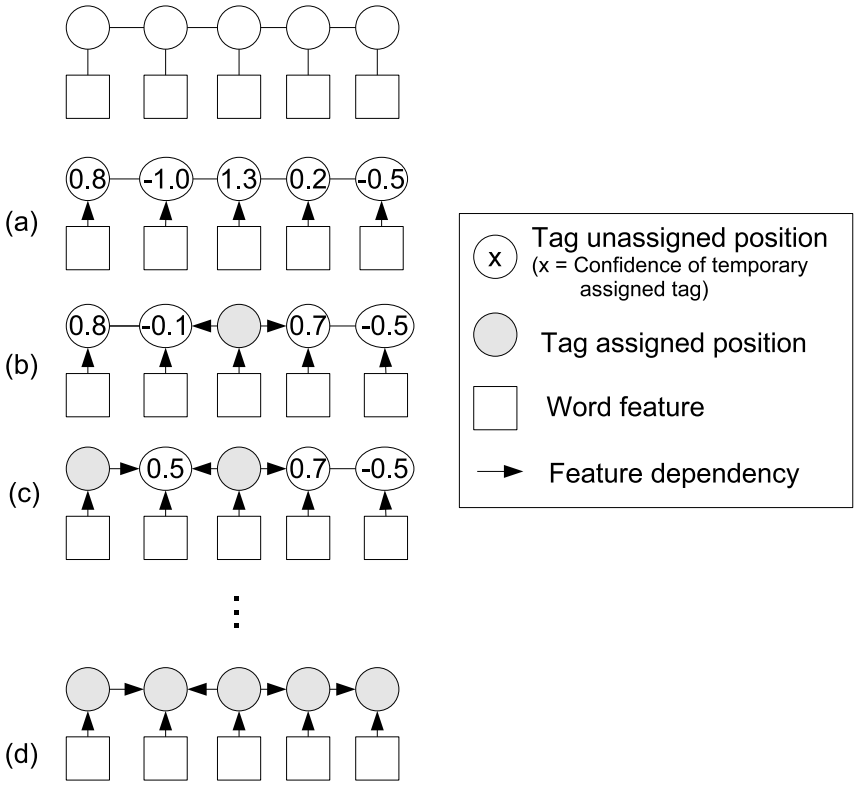


Figure 4.2. Easiest-first sequential model

The disadvantage of sequential classification models is that they cannot redo tag prediction once the tag is predicted, i.e, the models cannot exploit (probably) useful tag information that may come later. This may cause a serious bias that tags which occur frequently with the already assigned tags are preferred than tags which occur frequently with to-be-assigned tags. This bias is named "label bias" by Lafferty *et al.* [32]

Some authors address this label bias problem by applying a beam search to delay predictions[27, 28] although Ratnaparki does not report its effectiveness in [27], and Kudo *et al.* report it does not result in any significant improvements[29].

2.2 One-shot Prediction Models

One-shot prediction models [33, 32, 34, 35] searches the best *tag sequence* using a scoring function of a whole sequence. The scoring function does not evaluate any single tag in isolation as is the case in sequential classification models.

Different one-shot prediction models use different types of scoring functions. Hidden Markov Models [33] assume a stochastic generative process of tagged sequence in which tags are hidden states and words are observable outputs emitted from hidden states. The score of a tag sequence is computed from the probabilities of state transition and emission, which are estimated from training data. Conditional Random Fields (CRFs) [32] treat sequence tagging as an inference problem of maximum likelihood configuration on a Markov random field over a chain. The likelihood of a tag sequence (i.e, a configuration) is computed as a product of clique potentials, which is estimated from training data using log-linear models. Hidden Markov Support Vector Machines [34] and Max-Margin Markov Networks [35] use a linear combination of tagged sequence features as a scoring function. The weights are optimized so that the score of correctly tagged sequence becomes as much higher than those of wrongly tagged sequence as possible.

An obvious advantage of one-shot prediction models is that they do not suffer from label bias since they select the highest scored tag sequence from all possible tag sequences. In fact state-of-the-art one-shot prediction models like CRFs often outperform sequential classification models. Searching the highest scored tag sequence, however, is computationally demanding and one-shot prediction models usually run much slower than sequential classification models.

3. Problem Statement

Sequential classification models run much faster than one-shot prediction models, but suffer from label bias. Thus one of natural research directions is to develop a method to make label bias less serious in sequential classification models while keeping computational costs low.

The easiest-first model [31] can be considered as one of such attempts. It assigns tags to easy-to-predict parts of sequences before hard-to-predict parts. Thus when the hard parts are tagged, it is likely that tag classifier can use adjacent and (probably) reliable tag information. Although this does not completely eliminate label bias, the situation is much better than fixed-order sequential models such as left-to-right and right-to-left models.

The easiest-first model determines next tagging positions solely on the basis of tag classifier's confidence score. However it is not clear that classifier's score is a good indicator of the reliability of classifier's prediction. In addition to that, no linguistic information is used to determine next positions although some linguistic information is definitely helpful to know easy-to-predict parts in sequences. (For example in named entity recognition, if the word "Bush" appears just after "President", it is almost certain that this "Bush" means the name of a person, not a shrub.)

From the above considerations, the following questions arise.

- *How should classifier's confidence score and linguistic information be combined to guess a good tagging position?*
- *How much improvement can be expected if the optimal tagging positions are chosen in sequential classification models?*

The second question is as important as the first one because an answer to the second question should give an upper bound to the impact of guessing good tagging positions. Therefore we will answer the second question first.

4. Optimal Sequential Classification

From here, $\mathbf{x}$ and $\mathbf{y}$ represent a word sequence and its tag sequence respectively. The i-th word (tag) of $\mathbf{x}(\mathbf{y})$ is resented by x_i (y_i). T denotes the set of all tags. Additionally we assume that the following four tag classifiers are given. (Figure 4.3)

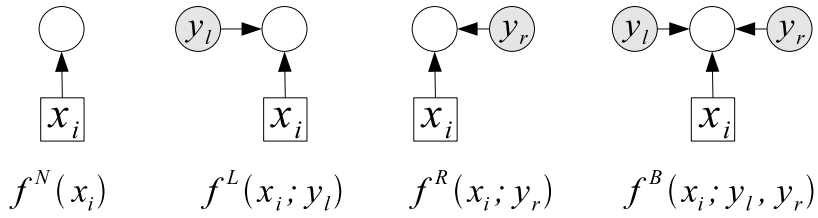


Figure 4.3. Tag classifiers

- $f^N(x_i)$ predicts the i-th word's tag given only the feature of i-th word.
- $f^L(x_i; y_l)$ predicts the i-th word's tag given the feature of i-th word and the left-side tag y_l.
- $f^R(x_i; y_r)$ predicts the i-th word's tag given the feature of i-th word and the right-side tag y_r.
- $f^B(x_i; y_l, y_r)$ predicts the i-th word's tag given the feature of i-th word, the left-side tag y_l and the right-side tag y_r.

All methods and discussions in this thesis can be easily extended to use more complex tag classifiers such as the one using second left-side tag as a feature. We, however, decided to use only these four tag classifiers in this thesis because the addition of tag classifiers makes algorithms more computationally demanding and it becomes very difficult to repeat experiments.

To make statements simpler, we introduce a concept *strategy*. *Strategy* is the ordering of tagging positions. More formally speaking, a strategy for $\mathbf{x}$ is a permutation $\langle i_1, i_2, \ldots, i_n \rangle$, where n is the number of words in $\mathbf{x}$, $1 \leq i_k, i_l \leq n$ and $i_k \neq i_l$ for $\forall k \neq l$. If a strategy $\langle i_1, i_2, \ldots, i_n \rangle$ is given, one can stepwise predict the tags $\langle y_{i_1}, y_{i_2}, \ldots, y_{i_n} \rangle$ using f^N (if both-side words are not tagged), f^L (if only left-side word is tagged), f^R (if only right-side word is tagged), or f^B (if both-side words are tagged) for each step prediction.

With these notations, the second question of Sec.3 can be restated as follows.

> *Given* $\mathbf{x}$ *and* $\mathbf{y}$*, find the optimal strategy which minimize the tag error of the resultant tag sequence.*

Here "tag error" means the number of wrong tag assignments.

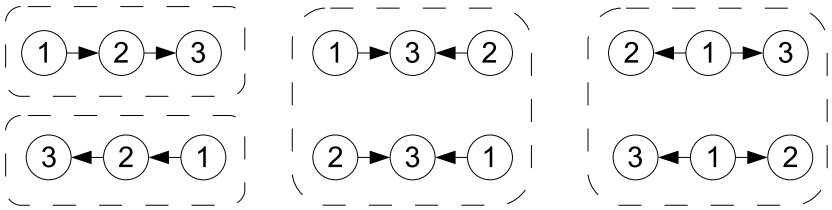


Figure 4.4. All possible strategies for a 3-word sequence. Six (= 3!) different strategies result in at most four (= 2^{3-1}) different tag sequences. Tag sequences with the same dependency are enclosed by a dashed lines.

4.1 Dynamic Programming for Minimum Error Strategy

To keep statements simple, we call tagging order between adjacent tags "dependency" hereafter. We say "a tag depends on its left (right) tag" if the tag is predicted after the left (right) tag.

Since a strategy corresponds to a permutation of n positions, there are $n!$ possible strategies. Thus it is practically impossible to enumerate all possible strategies and compare their errors in a brute force fashion. Fortunately, however, there is a dynamic programming method which finds the optimal strategy in a polynomial time complexity of n.

The dynamic programming method is based on the observation that output tags are completely determined by the dependencies between words, and some strategies result in identical dependencies. (Figure 4.4) This leads to the following restatement of the second question in Sec.3.

> *Given* $\mathbf{x}$ *and* $\mathbf{y}$*, find the optimal dependencies which minimize the tag error of the resultant tag sequence.*

To answer this question, we introduce the following variable.

$$\alpha^i_{t_i,d_i,t_{i+1}} \equiv \min_{t_j \in T, d_j \in D (1 \le j \le i-1)} \sum_{k=1}^{i} \mathrm{I}[y_k \neq t_k | t_k = f(x_k; t_{k-1}, d_{k-1}, d_{k+1}, t_{k+1})]. \quad (4.1)$$

Here T is the set of all possible tags, D the set of two dependencies $\{\leftarrow, \rightarrow\}$ ($\leftarrow$ ($\rightarrow$) means that the left(right)-side tag depends on the right(left)-side tag); t_i is the predicted tag of the i-th word and d_i is the dependency between i-th and

$(i+1)$-th tags; $\mathrm{I}[\cdots|\cdots]$ is a conditional indicator function,

$$\mathrm{I}[\mathit{statement}|\mathit{condition}] = \begin{cases} 1 & \text{if } \mathit{condition} \text{ is true} \wedge \mathit{statement} \text{ is true} \\ 0 & \text{if } \mathit{condition} \text{ is true} \wedge \mathit{statement} \text{ is false} \\ \infty & \text{if } \mathit{condition} \text{ is false,} \end{cases} \quad (4.2)$$

and f is a proxy tag classifier defined as

$$f(x; t_l, d_l, d_r, t_r) \equiv \begin{cases} f^N(x) & \text{if } d_l = \leftarrow \wedge d_r = \rightarrow \\ f^L(x; t_l) & \text{if } d_l = \rightarrow \wedge d_r = \rightarrow \\ f^R(x; t_r) & \text{if } d_l = \leftarrow \wedge d_r = \leftarrow \\ f^B(x; t_l, t_r) & \text{if } d_l = \rightarrow \wedge d_r = \leftarrow . \end{cases} \quad (4.3)$$

Since the k-th summand in Eq.(4.1) is the tag error of the k-th word, the minimum error is equal to $\min_{t_n} \alpha^n_{t_n,\leftarrow,\mathrm{EOS}}$, where EOS is the special tag for the end of sequence [2].

$\alpha^n_{t_n,\leftarrow,\mathrm{EOS}}$ can be evaluated in $O(n)$ using the following recursion.

$$\alpha^i_{t_i,d_i,t_{i+1}} = \min_{t_{i-1},d_{i-1}} \alpha^{i-1}_{t_{i-1},d_{i-1}i,t_i} + \mathrm{I}[y_i \neq t_i | t_i = f(x_i; t_{i-1}, d_{i-1}, d_{i+1}, t_{i+1})] \quad (4.4)$$

$$\alpha^0_{t_0,d_0,t_1} = \begin{cases} 0 & \text{if } t_0 = \mathrm{BOS} \wedge d_0 = \rightarrow \\ \infty & \text{otherwise} \end{cases} \quad (4.5)$$

Figure4.5 is a pseudo code for computing α and minimum tagging error. We separate the main part of computation into function MinErrorTagging which outputs the minimum error tag sequence instead of the minimum error itself. In MinErrorTagging, variable $A^{(k)}_{t_k,d_k,t_{k+1}}$ keeps track of a part of minimum error tag sequence which are read out at the last stage of the function. Please note that the algorithm in Figure4.5 can be seen as an adaptation of the bidirectional inference algorithms in [31], where the objective function is a total confidence of tag classifiers.

4.2 Impact of Minimum Error Strategy

To investigate how good minimum error tag sequence is, We computed minimum tagging error on real NLP tagging data and compared it to some sequential classification models.

[2] We assume that virtual tags BOS(beginning of sequence) and EOS are assigned to both sides of sequence before tagging, and tag classifiers accept these virtual tags as feature.

```
1 :  function MinError(x, y)
2 :  n := |x|
3 :  ŷ := MinErrorTagging(x, y, 1, n, BOS, EOS)
4 :  err := (1/n) Σ_{i=1}^{n} I[y_i ≠ ŷ_i]
5 :  return err

1 :  function MinErrorTagging(x, y, s, e, t_{s-1}, t_{e+1})
2 :  // s(e) : Start(End) index to tag
3 :  // t_{s-1}(t_{e+1}) : Tag of x_{s-1}(x_{e+1})
4 :  α := ∞
5 :  for each t ∈ T
6 :     α_{t_{s-1},→,t} := 0
7 :  // Main loop
8 :  for i := s ... e
9 :     if i < e    D' := D, T' := T
10 :    else        D' := {←}, T' := {t_{e+1}}
11 :    α' := ∞
12 :    for each d_r ∈ D', t_r ∈ T'
13 :       for each ⟨t_l, d_l, t⟩ where α_{t_l,d_l,t} ≠ ∞
14 :          if α'_{t,d_r,t_r} > α_{t_l,d_l,t} + I[y_i ≠ t|t = f(x_i; t_l, d_l, d_r, t_r)]
15 :             α'_{t,d_r,t_r} := α_{t_l,d_l,t} + I[y_i ≠ t|t = f(x_i; t_l, d_l, d_r, t_r)]
16 :             A^{(i)}_{t,d_r,t_r} := ⟨t_l, d_l⟩
17 :    α := α'
18 :  // Read out the minimum error path
19 :  d_r :=←, ŷ_{e+1} := t_{e+1} , ŷ_e := arg min_{t∈T} α_{t,d_r,ŷ_{e+1}}
20 :  for i := e ... s
21 :     ⟨ŷ_{i-1}, d_l⟩ := A^{(i)}_{ŷ_i,d_r,ŷ_{i+1}}
22 :     d_r := d_l
23 :
24 :  return ŷ_{s,e}
```

Figure 4.5. Pseudo Code of Minimum Error Tagging

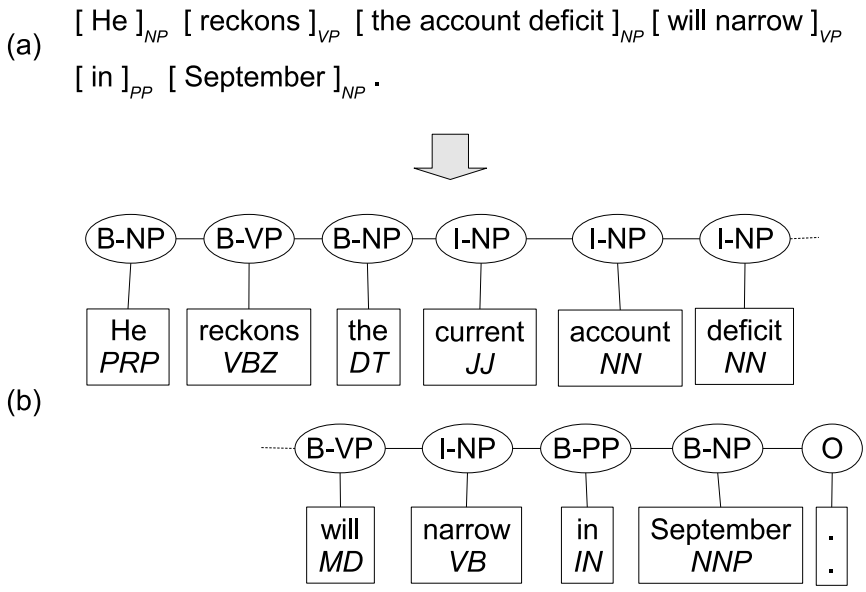


Figure 4.6. An Example of Chunking Data. (a) A sentence with chunking (phrase) information and (b) its converted form for tagging.

Data

We used CoNLL-2000 Chunking dataset [36] for this purpose. The dataset consists of 8,936 training sentences and 2,012 test sentences which are from news articles of Wall Street Journal. Each word in sentences are annotated with a part-of-speech and a chunking tag. (Figure 4.6) A chunking tag represents that the word is the beginning of phrase (B-*), the inside or end of phrase (I-*), or does not belong to any phrase (O).

Tag Classifiers

One-vs-rest Support Vector Machines with polynomial kernels were used as tag classifiers. We used a binary feature vector, whose element is associated with a unique feature pattern. An element is equal to 1 if its associated pattern matches the context of x_i and 0 otherwise. The following templates are used to generate the feature patterns.

- x_k's token is [TKN]. ($i-2 \leq k \leq i+2$)
- x_k's part-of-speech is [POS]. ($i-2 \leq k \leq i+2$)
- (Only for f^L and f^B) y_{i-1} is [TAG].
- (Only for f^R and f^B) y_{i+1} is [TAG].

A feature pattern is generated by replacing [TKN], [POS] and [TAG] by any token, part-of-speech and tag respectively. For example, Table 4.1 shows the feature patterns that match the context of the word "current" in Figure 4.6.

The degree of polynomial kernel d and the hyper-parameter C are optimized by 10-fold cross validation on training data. We tested all combination $d = 1, 2, 3$ and $C = 0.001, 0.01, 0.1, 1.0$, then chose $d = 2$ and $C = 0.1$.

Results

Table 4.2 shows the accuracy and F-measure of minimum tag error sequence, three sequential classification models (left-to-right, right-to-left, easiest-first) and the best three results reported in the CoNLL-2000 Shared Task page [3] (Zhang *et al.*, Kudo *et al.*, Carreras *et al.*).

[3] http://www.cnts.ua.ac.be/conll2000/chunking/

x_{i-2}'s token is "reckons".
x_{i-1}'s token is "the".
x_i's token is "current".
x_{i+1}'s token is "account".
x_{i+2}'s token is "deficit".
x_{i-2}'s part-of-speech is VBZ.
x_{i-1}'s part-of-speech is DT.
x_i's part-of-speech is JJ.
x_{i+1}'s part-of-speech is NN.
x_{i+2}'s part-of-speech is NN.
(Only for f^L and f^B) y_{i-1} is B-NP.
(Only for f^R and f^B) y_{i+1} is I-NP.

Table 4.1. The feature patterns which match the context of "current" in Fig. 4.6.

	Accuracy	F-measure
Left-to-right	95.85	93.56
Right-to-left	95.90	93.66
Easiest-first	95.98	93.73
Zhang *et al.*[37]	—	94.13
Kudo *et al.*[29]	—	93.91
Carreras *et al.*[38]	—	93.74
Min. Tag Error	**97.02**	**95.18**

Table 4.2. Performance of Minimum Error Strategy

Accuracy and F-measure are defined as

$$\text{Accuracy} \equiv \frac{(\#\text{ of Correctly Predicted Tags})}{(\#\text{ of Tags})}, \tag{4.6}$$

$$\text{F-measure} \equiv \frac{2 \times (\#\text{ of Correctly Recognized Chunks})}{(\#\text{ of True Chunks}) + (\#\text{ of Recognized Chunks})}. \tag{4.7}$$

Table 4.2 clearly shows that the minimum error tag sequences are much better than not only the sequential classification methods, but also the state-of-the-art methods. The improvement is impressive: the minimum error results are better than the top result by around 1 point in F-measure, whereas the difference in F-measure between the top result and the (rather naive) left-to-right result is only 0.57 point.

Thus the second question in Sec.3 has been answered quite positively: much better tagging is achieved if we know better strategies. That also means that the first question in Sec.3 is indeed worth to consider.

5. Learning of Tagging Strategy

First we introduce some notations and concepts.

$\mathbf{x}$ and $\mathbf{y}$ are the same as in the previous section. $\hat{y}_i \in T \cup \{\emptyset\}$ is a tag assigned to the i-th word by a tag classifier, where $\hat{y}_i = \emptyset$ means that the i-th word is not assigned a tag yet. We call a pair $(\mathbf{x}, \hat{\mathbf{y}})$ "a partially tagged sequence." A strategy function $g(\mathbf{x}, \hat{\mathbf{y}})$ is a map from partially tagged sequences to untagged positions in $\hat{\mathbf{y}}$, i.e, $\{k|\hat{y}_k = \emptyset\}$.

With the above notations, the first question in Sec.3 can be restated as follows.

> *How to construct a good strategy function from classifier's confidence score and linguistic information?*

In this thesis, we will investigate a machine learning approach to construct a good strategy function. For brevity, we simply call learning of strategy function "strategy learning" hereafter.

5.1 Generating Training Data

In this thesis, we assume that sample tagged sequences are given to the learner. These sequences, however, cannot be used directly as the training data for strategy learning because the training data should be pairs of strategy function's input

and output, i.e., partially tagged sequence and next tagging positions. Thus it is necessary to generate training data from given tagged sequences.

Concerning training data generation, there are two problems.

1. Which partially tagged sequences should be selected as input samples?
2. How do we define "good" tagging positions?

Selection of Partially Tagged Sequences

The number of possible partially tagged sequences is so huge that it is impossible to use all of them as training data. In addition to that, it is useless (and probably harmful) to include the partially tagged sequences which would not appear during tagging.

Based on the above discussion, we propose to use partially tagged sequences which appear during tagging along the current strategy. Figure 4.7 is a pseudo code of the proposed training data generation process. In the algorithm, we use proxy tag classifier f [4],

$$f(x; y_l, y_r) \equiv \begin{cases} f^N(x) & \text{if } y_l = \emptyset \wedge y_r = \emptyset \\ f^L(x; y_l) & \text{if } y_l \neq \emptyset \wedge y_r = \emptyset \\ f^R(x; y_r) & \text{if } y_l = \emptyset \wedge y_r \neq \emptyset \\ f^B(x; y_l, y_r) & \text{if } y_l \neq \emptyset \wedge y_r \neq \emptyset. \end{cases} \tag{4.8}$$

For each sequence the algorithm starts with InitialTagging. Function InitialTagging assigns highly confident tags before tagging with the current strategy g. This greatly reduces the run time of data generation without deteriorating the quality of data. (See the experimental results for detail.) Then the algorithm generates a training sample $(\mathbf{x}, \hat{\mathbf{y}}, i)$ (i.e, partially tagged sequence and next tagging position) in each tagging step with the given strategy g (Line 7-14). To generate the sample, GenerateTrainData calls function NextTaggingPosition, which estimates the best tagging position for the next step. We will explain NextTaggingPosition below.

Evaluation of Tagging Position

Intuitively an untagged position should be considered as a good tagging position when an accurate tag sequence is obtained if the untagged position is tagged

[4]Here we reuse the notation f, which is also used in Eq.(4.3), since both f are proxy tag classifiers and it is obvious which f is used from context.

```
1 : function GenerateTrainData(g, {(x^k, y^k)}_{k=1}^m)
2 : // g : Strategy function
3 : // {(x^k, y^k)}_{k=1}^m : Samples of tagged sequence
4 : D := {}
5 : for each (x, y) ∈ {(x^k, y^k)}_{k=1}^m
6 :       ŷ := InitialTagging(x)
7 :       U := {i|ŷ_i = ∅}
8 :       while U ≠ {}
9 :             // Generate training data
10 :            i := NextTaggingPosition(x, ŷ, y)
11 :            Add (x, ŷ, i) to D
12 :            // Tagging with the current strategy
13 :            i := g(x, ŷ)
14 :            ŷ_i := f(x_i; ŷ_{i-1}, ŷ_{i+1})
15 :            U := U/{i}
16 : return D
```

```
1 : function InitialTagging(x)
2 : for i := 1, ..., |x|
3 :       if the confidence score of f^N(x_i) > given threshold
4 :             y_i := f^N(x_i)
5 :       else
6 :             y_i := ∅
7 : return ŷ
```

Figure 4.7. Pseudo code of training data generation

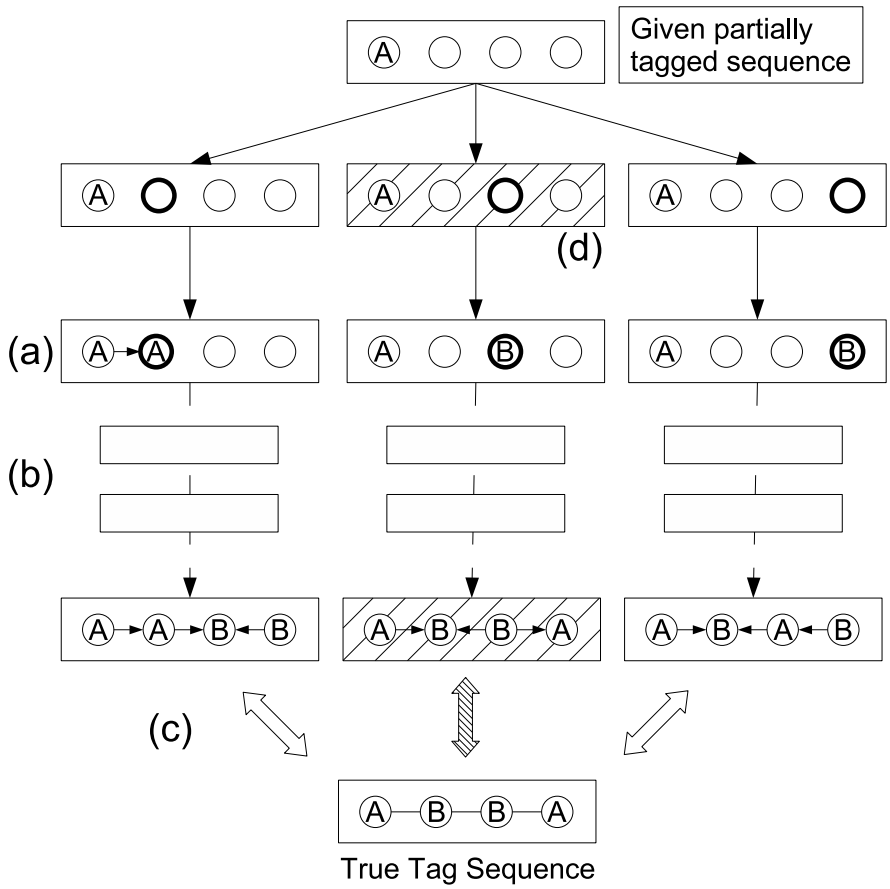


Figure 4.8. Illustration of evaluation process of next tagging position (a) First each candidate position is tagged. (b) Remaining empty position is filled by the evaluation strategy. (c) The result sequence is evaluated using true tag sequences. (d) The candidate position from which the best result originated is choosen as the best position to tag in the next step.

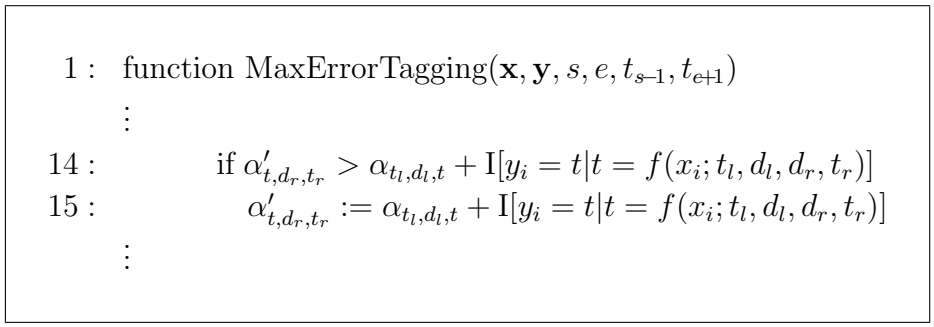


Figure 4.9. Pseudo Code of Maximum Error Tagging. Line 2-13 and 16-24 are the same as MinErrorTagging, thus omitted.

first and then the remaining positions are tagged along a strategy function the learner will finally produce. (Figure 4.8) This characterization of good tagging position, however, involves a circular argument since it uses the to-be-learned strategy to learn the strategy itself. To circumvent this problem, we propose to use one of the following strategies instead of the to-be-learned strategy. (We call these strategies "evaluation strategies" to emphasize that they are only used to "evaluate" the goodness of next tagging position.)

Minimum Error Strategy Assign tags to untagged words using MinErrorTagging in Figure 4.5. This strategy produces the most accurate tag sequence which can be derived from given partially tagged sequence, thus gives an upper bound of the position's goodness. This strategy may not be learnable.

Current strategy Assign tags using the strategy being learned. This strategy is not so optimistic as Minimum Error Strategy and is learnable by definition.

Maximum Error Strategy Assign tags to untagged words so that the resultant tag sequence becomes the most *inaccurate* tag sequence. Figure 4.9 shows an algorithm which performs (sub)sequence tagging with this strategy. On the contrary to Minimum Error Strategy, Maximum Error Strategy evaluates tagging positions in their worst case.

Figure 4.10 shows pseudo codes of NextTaggingPosition and the above evaluation strategies (EvalTagging_*).

```
1 : function NextTaggingPosition(x, ŷ, y)
2 :  n := |x|
3 :  U := {i|ŷ_i = ∅}
4 :  for each i ∈ U
5 :       ŷ_i := f_i(x, ŷ)
6 :       ŷ := EvalTagging(x, ŷ, y)
7 :       l_i = (1/n) Σ_{i=1}^{n} I[y_i ≠ ŷ_i]
8 :  return arg min_{i∈U} l_i

1 : // Evaluated with Minimum Erorr Strategy
2 : function EvalTagging_MinError(x, ŷ, y)
3 :  R := {(s, e)|1 ≤ s < e ≤ n ∧ ŷ_{s−1} ≠ ∅ ∧ ŷ_{e+1} ≠ ∅ ∧ ŷ_i = ∅ for ∀i ∈ [s, e]}
4 :  for each (s, e) ∈ R
5 :       ŷ_{s,e} := MinErrorTagging(x, y, s, e, ŷ_{s−1}, ŷ_{e+1})
6 :  return ŷ

1 : // Evaluated with Maximum Error Strategy
2 : function EvalTagging_MaxError(x, ŷ, y)
3 :  R := {(s, e)|1 ≤ s < e ≤ n ∧ ŷ_{s−1} ≠ ∅ ∧ ŷ_{e+1} ≠ ∅ ∧ ŷ_i = ∅ for ∀i ∈ [s, e]}
4 :  for each (s, e) ∈ R
5 :       ŷ_{s,e} := MaxErrorTagging(x, y, s, e, ŷ_{s−1}, ŷ_{e+1})
6 :  return ŷ

1 : // Evaluated with the current strategy g
2 : function EvalTagging_CurrentStrategy(x, ŷ, y)
3 :  U := {n|ŷ_n = ∅}
4 :  while U ≠ {}
5 :       i := g(x, ŷ)
6 :       ŷ_i := f(x_i; ŷ_{i−1}, ŷ_{i+1})
7 :       U := U/{i}
8 :  return ŷ
```

Figure 4.10. Pseudo code of position evaluation

5.2 Learning of Linear Strategy Model

We described a generation method of training data. Now we turn to the problem of how to learn a strategy function from generated training data.

A Linear Strategy Model

We assume that there are position scoring functions $h_i(\mathbf{x}, \mathbf{y})$ behind a strategy function and the strategy function returns the highest score position, i.e,

$$g(\mathbf{x}, \hat{\mathbf{y}}) = \arg \max_{i \in \{j | \hat{y}_j = \emptyset\}} h_i(\mathbf{x}, \mathbf{y}). \tag{4.9}$$

Additionally it is assumed that the position scoring function is a linear function of a position-dependent feature vector $\vec{\phi}_i(\mathbf{x}, \hat{\mathbf{y}})$,

$$h_i(\mathbf{x}, \mathbf{y}) = \vec{w} \cdot \vec{\phi}_i(\mathbf{x}, \hat{\mathbf{y}}), \tag{4.10}$$

where $\vec{w}$ is a weight vector. Here are some examples of position-dependent features.

- Information of i-th word and its neighborhood words
- Already assigned tags around i-th word
- Predicted tag and its confidence score by tag classifier

We show a pseudo code of sequence tagging with the above linear model in Figure4.11.

In Figure 4.11, tagging a whole sequence needs n (n is the length of the input sequence $\mathbf{x}$) iterations of the loop, and the k-th iteration involves the search of the maximum element among $n-k-1$ untagged positions. (Line 4 in Figure 4.11) Thus it takes $O(n^2)$ computational steps in general to assign all tags. This $O(n^2)$ computational cost is against our research motivation of improving sequential classification models while keeping computational cost low. However we can reduce this cost to $O(n \log n)$ if the position-dependent feature vector $\vec{\phi}_i(\mathbf{x}, \hat{\mathbf{y}})$ is designed so that a tag assignment to the i-th word results in a limited number of other feature vectors $\vec{\phi}_{j \neq i}(\mathbf{x}, \hat{\mathbf{y}})$ [31].

To explain how this reduction can be done, suppose that a tag assignment affects at most k feature vectors. First we store the scores $\{\vec{w} \cdot \vec{\phi}_j(\mathbf{x}, \hat{\mathbf{y}})\}_{j \in U}$ in a red-black tree [39]. This step takes at most $O(n \log n)$. Then in each iteration

```
1 : function SequenceTagging(x)
2 : U := {i|ŷ_i = ∅}
3 : while U ≠ {}
4 :       i := arg max_{j∈U} w⃗ · φ⃗_j(x, ŷ)
5 :       ŷ_i := f(x_i; ŷ_{i-1}, ŷ_{i+1})
6 :       U := U/{i}
7 :
8 : return ŷ
```

Figure 4.11. Pseudo code of sequence tagging with linear strategy model

```
1 : function UpdateStrategy(w⃗, {(x^k, ŷ^k, i^k)}_{k=1}^m)
2 : for n := 1, 2, ..., N
3 :       for each (x, y, i) ∈ {(x^k, ŷ^k, i^k)}_{k=1}^m
4 :             U := {j|ŷ_j = ∅}
5 :             for j ∈ U/{i}
6 :                   if   w⃗ · φ⃗_i(x, ŷ) ≤ w⃗ · φ⃗_j(x, ŷ)
7 :                         w⃗ = w⃗ + (φ⃗_i(x, ŷ) − φ⃗_j(x, ŷ))
8 : return w⃗
```

Figure 4.12. Pseudo code of strategy update

in Figure 4.11, we search the highest score position (this can be done within $O(\log n)$) and update (i.e, delete and insert) at most k scores (this can be done within $O(k \log n)$). Thus the whole tagging can be done in $O(kn \log n)$.

Perceptron-based Update

Figure 4.12 shows the learning algorithm for our linear strategy models. The algorithm is an adaptation of Perceptron variants proposed by Collins *et al.* [40, 41, 42]. The algorithm updates the weight vector $\vec{w}$ only when the score of the to-be-tagged position $\vec{w} \cdot \phi_i(\mathbf{x}, \hat{\mathbf{y}})$ is lower than those of the other positions. The update is done so that to-be-tagged position scores get higher while the other position scores lower. $\vec{w}$ converges within a finite iterations if the generated

```
1 : function StrategyLearn(\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k=1}^m)
2 : Initialize \vec{w}
3 : \vec{u} := \vec{w}
4 : for t := 1, 2, \ldots, T
5 :       D := GenerateTrainData(g, \{(\mathbf{x}^k, \mathbf{y}^k)\}_{k=1}^m)
6 :       UpdateStrategy(\vec{w}, D)
7 :       \vec{u} := \vec{u} + \vec{w}
8 : return \vec{u}/T
```

Figure 4.13. Pseudo code of strategy learning

training data are "separable" i.e, there is a weight vector which can score all the to-be-tagged position higher than the others. [42].

5.3 Full Description of Strategy Learning

Figure 4.13 shows the main algorithm of the proposed strategy learning method. To avoid over-fitting and enhance generalization, StrategyLearn outputs not the weight vector $\vec{w}$, but the averaged weight vector $\vec{u}/T$. This type of averaging is investigated in [43] and the experimental results there shows constantly better results than the non-averaged case.

5.4 Round Robin Training

When we use a good machine learning algorithm to construct tag classifiers, it is better to split training data for tag classifiers learning and strategy function learning. This is because tag classifiers are generally so good at predicting tags on training data that any tagging strategy (even a random strategy) could produce quite accurate tagging. Thus it is very hard to learn a meaningful strategy from the training data for tag classifiers. Splitting training data, however, also means that neither tag classifier learning algorithm or strategy learning algorithm can use the whole training data. This usually results in inaccurate tag classifier and strategy functions.

To overcome this trade-off, we propose *Round Robin Training*. (Figure 4.14) Round Robin Training proceeds as follows.

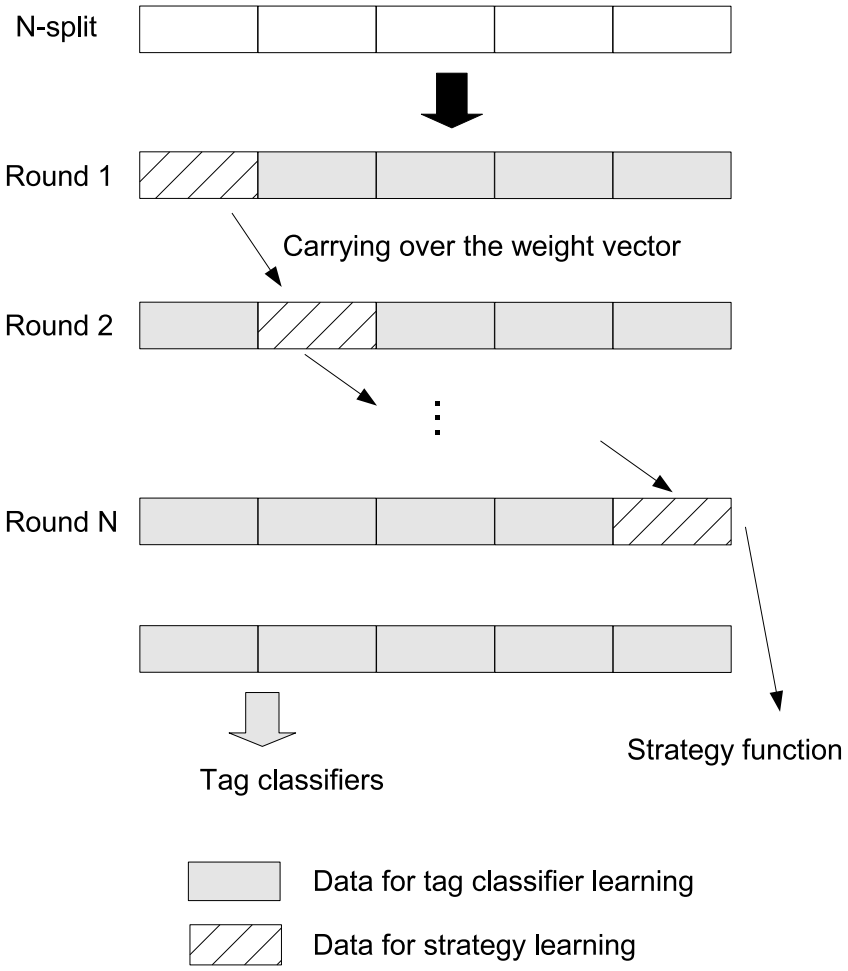


Figure 4.14. Round robin training

1. Split given training data into N parts of the same size.
2. Repeat step 3 and 4 from $i = 1$ to N
3. Learn tag classifiers using the training data except the i-th part.
4. Learn strategy function with the tag classifiers learned in step 3 using the i-th part as training data. Use the weight vector in the previous iteration as the initial value of w.
5. Output the averaged weight vector as the parameter of strategy function.
6. Learn tag classifiers using all the training data.

Training data for tag classifiers and strategy function do not overlap in each iteration of Round Robin Training. Additionally both tag classifiers and strategy function are trained using all the training data. Thus Round Robin Training avoids both problems described at the beginning of this section.

6. Experiments

6.1 Data and Tag Classifiers

We used the same CoNLL-2000 Chunking data and the same learning conditions of tag classifiers (i.e, feature, hyper-parameter, and kernel) as in Sec.4.2.

6.2 Strategy Learning Setup

We use feature vector $\vec{\phi}_i(\mathbf{x}, \hat{\mathbf{y}})$ which consists of a real-valued element and binary-valued elements. The real-valued element is the confidence score of $f(x_i; \hat{y}_{i-1}, \hat{y}_{i+1})$, i.e, the output of the SVM which corresponds to $f(x_i; \hat{y}_{i-1}, \hat{y}_{i+1})$. The binary elements represent the existence of feature patterns generated from the following feature templates.

- x_k's token is [FREQTKN]. ($i-1 \leq k \leq i+1$)
- x_k's part-of-speech is [POS] ($i-1 \leq k \leq i+1$).
- The pair of x_k and x_l's tokens is ([FREQTKN1], [FREQTKN2]). ($i-1 \leq k < l \leq i+1$)

- The pair of x_k and x_l's part-of-speeches is ([POS1], [POS2]). ($i-1 \leq k < l \leq i+1$)
- The pair of x_k's token and x_l's part-of-speech is ([FREQTKN], [POS]). ($i-1 \leq k < l \leq i+1$)
- The pair of x_k's token and $f(x_i; \hat{y}_{i-1}, \hat{y}_{i+1})$ is ([FREQTKN], [TAG]). ($i-1 \leq k \leq i+1$)
- The pair of x_k's part-of-speech and $f(x_i; \hat{y}_{i-1}, \hat{y}_{i+1})$ is ([POS], [TAG]). ($i-1 \leq k \leq i+1$)
- A part-of-speech trigram within a chunk after tagging is [CHUNK_POS_TRIGRAM].

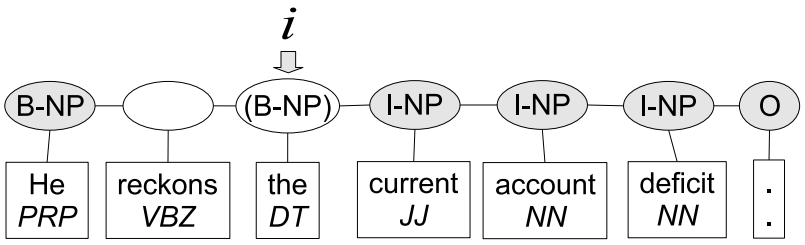
Here [FREQTKN*] represents a token which is among 200 most frequent tokens in the training data. [CHUNK_POS_TRIGRAM] represents a trigram of part-of-speeches which does not cross chunk boundaries. An example of feature patterns is shown in Table 4.3.

6.3 Results

Effects of Learned Strategy

Figure 4.15 and Figure 4.16 show the accuracies and the F-measures of learned strategy tagging on the CoNLL-200 test sentences. The tag classifiers were trained on 1936 sentences randomly drawn from the CoNLL-2000 training sentences, and the strategy function were trained on 1000, 2000, 3000, 4000, 5000, 6000, 7000 sentences randomly drawn from the remaining training sentences. We did not use Round Robin Training in this experiment to see the genuine difference between learned and non-learned strategies. The threshold in InitialTagging was 1.0, the number of UpdateStrategy iterations (N in Figure 4.12) 5, and the number of StrategyLearn iterations (T in Figure 4.13) 10. Minimum Error Strategy was used as the evaluation strategy in Figure 4.10.

Figure 4.15 and Figure 4.16 show that the proposed strategy learning method produces better strategies than non-strategy-learning sequential classification methods (left-to-right, right-to-left and easiest-first). Additionally the improvement is getting larger when more training sequences are given to the learner.



x_i's token is "the".
x_{i-1}'s part-of-speech is VBZ. x_i's part-of-speech is DT. x_{i+1}'s part-of-speech is JJ.
The pair of x_{i-1} and x_i's part-of-speeches is (VBZ, DT). The pair of x_{i-1} and x_{i+1}'s part-of-speeches is (VBZ, JJ). The pair of x_i and x_{i+1}'s part-of-speeches is (DT, JJ).
The pair of x_i's token and x_{i-1}'s part-of-speech is ("the", VBZ). The pair of x_i's token and x_i's part-of-speech is ("the", VBZ). The pair of x_i's token and x_{i+1}'s part-of-speech is ("the", JJ).
The pair of x_i's token and $f(x_i; \hat{y}_{i-1}, \hat{y}_{i+1})$ is ("the", B-NP).
The pair of x_{i-1}'s part-of-speech and $f(x_i; \hat{y}_{i-1}, \hat{y}_{i+1})$ is (VBZ, B-NP). The pair of x_i's part-of-speech and $f(x_i; \hat{y}_{i-1}, \hat{y}_{i+1})$ is (DT, B-NP). The pair of x_{i+1}'s part-of-speech and $f(x_i; \hat{y}_{i-1}, \hat{y}_{i+1})$ is (JJ, B-NP).
A part-of-speech trigram within a chunk after tagging is NP:%BOC%-DT-JJ. A part-of-speech trigram within a chunk after tagging is NP:DT-JJ-NN. A part-of-speech trigram within a chunk after tagging is NP:NN-NN-%EOC%.

Table 4.3. Example feature patterns for strategy function. The feature patterns in the table match the i-th position in the partially tagged sequence at the top of the figure. The tokens "reckons" and "current" do not trigger any pattern because they are not frequent in training data. %BOC% represents a begining of chunk and %EOC% an end of chunk. The first NP chunk does not trigger any pattern because the end of the chunk can not be determined.

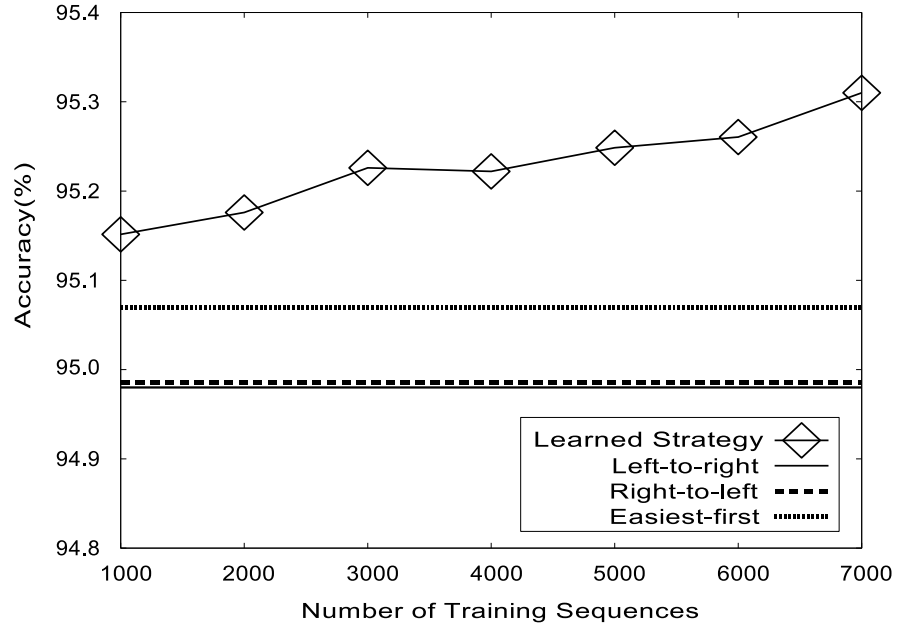


Figure 4.15. Learning curve of Accuracy

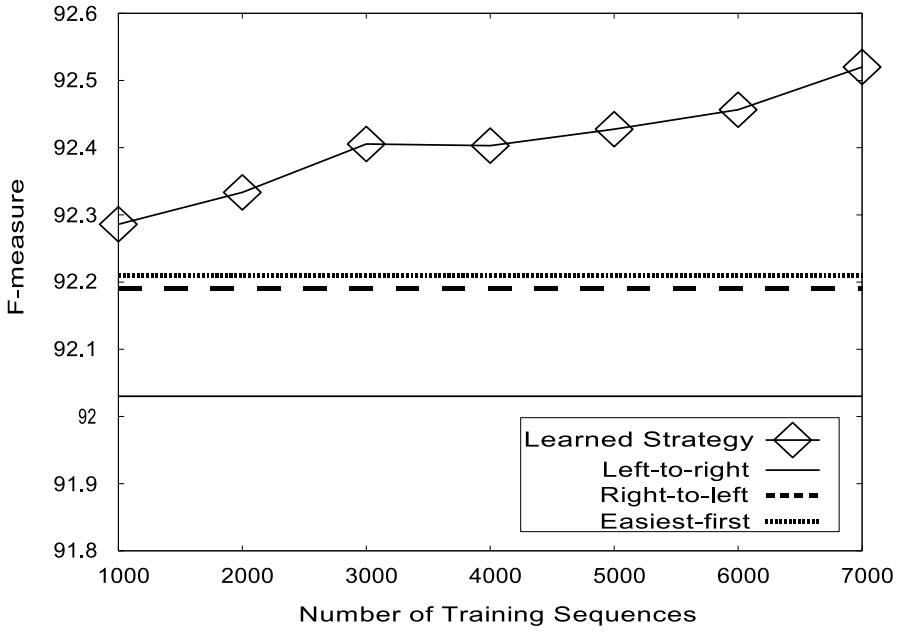


Figure 4.16. Learning curve of F-measure

Strategy	Accuracy	F-measure
Left-to-right	95.85	93.56
Right-to-left	95.90	93.66
Easiest-first	95.98	93.73
T20%-S80%	95.24	92.42
T40%-S60%	95.52	92.91
T60%-S40%	95.72	92.91
T80%-S20%	95.81	93.42
Round Robin	**96.10**	**93.89**

Table 4.4. Performance comparison of learning strategies (Txx%-Syy% and Round Robin) and non-learning strategies (Left-to-right, Right-to-left and Easiest-first).

Round Robin Training

One can say that the above experiment is unfair since the non-learned strategies uses only 1,936 sentences for training while the learned strategies uses additional 1,000-7,000 sentences for strategy training. Thus we conducted another experiment in which the whole training data is used to achieve best results for every method. Table 4.4 shows the accuracies and the F-measures of the learned and the non-learned strategies which were trained as follows. For non-learned strategies (i.e, left-to-right, right-to-left and easiest-first), tag classifiers are trained using the whole 8,936 training sentences. For learned strategies, two types of training were tested; (1) 20,40,60,80% of the training sentences were used for tag classifiers and the remains for strategy function (Txx%-Syy% in Table 4.4). (2) Round Robin Training with 5 splits (**Round Robin**). Other learning conditions were the same as in the first experiment.

In Table 4.4, accuracy and F-measure becomes better when the number of training data for strategy learning becomes *fewer*, whereas the previous experiment shows that the performance becomes better when the number of the training data becomes *larger*. (Figure 4.15 and Figure 4.16) This is because when training data for strategy learning gets smaller, training data *for tag classifiers* gets larger and tag classifiers become more accurate. This improvements in tag classifiers compensate the degradation in strategy function. Table 4.4 also indicates that Round Robin Training moderates such trade-off between tag classifiers and strat-

Evaluation Strategy	Acccuracy	F-measure
Minimum Error Strategy	**96.10**	**93.89**
Current Strategy	95.95	93.66
Maximum Error Strategy	95.90	93.56

Table 4.5. Performance comparison of different loss function and evaluation strategy combinations.

egy function and achieves much better results than the non-learning sequential tagging methods.

Evaluation Strategy

Table 4.5 shows the performances of different evaluation strategies. The learning conditions except evaluation strategy are the same as Round Robin Training in the previous experiment. The result of Minimum Error Strategy is much better than those of Current Strategy and Maximum Error Strategy.

From informal inspections of the learning process with Current Strategy, we observed that the convergence of the strategy function was much slower than the other strategies. This seems to be caused by the fact that the goodness of a tagging position change as learning proceeds and this makes it hard to learn a strategy function predicting the goodness. Minimum Error Strategy does not suffer from such value changes, since it is independent of strategy being learned,

Another inspection of Maximum Error Strategy shows another evaluation problem: Maximum Error Strategy does not tell good positions from bad ones in some cases where Minimum Error Strategy does. Figure 4.17 illustrates an example of such cases. In this example, we have a three-word sequence, whose possible tagging are shown in Figure 4.17(a). We assume that the bottom right tagging ("A-B-A") is correct. (Intuitively this means that the middle tag is hard to predict without the help of both sides' tag information.) Then Minimum Error Strategy can tell the learner that the right and the left are better positions than the middle. (Figure 4.17(b)) Maximum Error Strategy, however, will not show any preference between the three positions.(Figure 4.17(c)) Although this example is artificial, the essentially same situation, where there exists a position which is hard to predict without the help of both sides' tag, often occurs in real tagging problems.

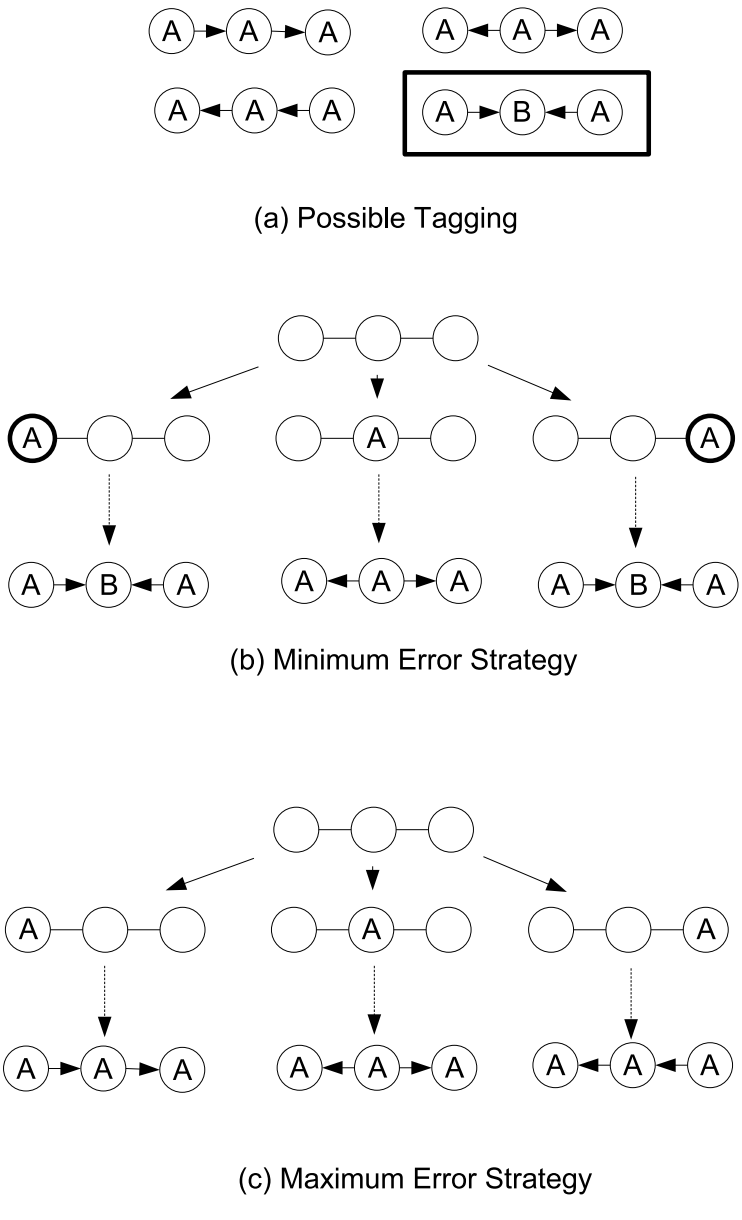


Figure 4.17. Comparison of Minimum Error Strategy and Maximum Error Strategy

Threshold	Accuracy	F-measure	Relative Training Time
0.0	95.92	93.38	0.2
0.2	95.99	93.82	0.3
0.4	96.04	93.79	0.3
0.6	96.05	93.81	0.6
0.8	96.07	93.81	0.7
1.0	**96.10**	**93.89**	**1.0**
1.2	96.09	93.89	3.2
1.4	96.03	93.78	10
1.6	96.04	93.81	24
1.8	96.02	93.79	34
2.0	96.05	93.82	53
∞	96.04	93.79	63

Table 4.6. Performance and runtime comparison of different thresholds for initial tagging. The relative training times are scaled so that the time of threshold 0.6 be 1.

Initial Tagging

Table 4.6 shows the performance measures (accuracy and F-measure) and the relative training time of strategies trained with different thresholds. The learning conditions except initial threshold are the same as Round Robin Training of the second experiment. It is observed that the performance is not so sensitive over a rather wide range of threshold although it decreases rapidly if the threshold is lowered too much. On the other hand the training time is very sensitive to the choice of threshold and can be much reduced while keeping high accuracy and F-measure.

7. Conclusion

In this chapter we addressed the problem of strategy learning in sequence tagging. First we presented a polynomial time algorithm for computing the minimum error strategy of a given tagged sequence and showed that tagging with the minimum error strategy produces very accurate tagging. Then a strategy learning algorithm were proposed. The experimental results showed that the learned

strategies achieves much higher performance than non-learning strategies.

Chapter 5

Conclusions

1. Summary of Thesis

In this thesis, we addressed three interdependent decision problems in natural language processing.

The first problem is sentence selection. (Chapter 2) We proved that a selection problem can be converted into a classification problem, thus can be solved by any classifier learning algorithm. Then Selection SVM is proposed to solve successive selection problems. Experiments with an artificial dataset and a sentence selection dataset show that Selection SVM performs better compared to SVMs and SVPL.

The second problem is multi-topic text categorization. (Chapter 3) We proposed a novel learning algorithm for multi-topic text categorization. The algorithm, Maximal Margin Labeling, embeds labels (i.e, sets of topics) into a similarity-induced vector space, and learns a large margin classifier in the space. To overcome the demanding computational cost of MML, we provide an approximation method in learning and efficient classification algorithms. In the experiments on a collection of Web pages, MML outperformed other methods such as SVM and nearest neighbor showed better generalization.

The last problem is sequence tagging. (Chapter 4) We proposed the strategy learning approach, whose goal is to learn a strategy function which guides where to tag in the next step. We presented a polynomial time algorithm for computing the minimum error strategy of a given tagged sequence and showed that tagging with the minimum error strategy leads to very accurate tagging. Then a strategy learning algorithm were proposed. The experimental results showed that the

learned strategies achieves much higher performance than non-learning strategies.

2. Implications for General Interdependent Decision Problems

The definition of interdependent decision problem is very general whereas this thesis covers only three examples of interdependent decision problems in NLP. In this section, we will discuss what implications this thesis' results have for general interdependent decision problems.

First we introduce some notations. $X_i(1 \leq i \leq m)$ is the i-th input variable and $Y_j(1 \leq j \leq n)$ the j-th output variable. When we mention the all input(output) variables at once, we use $X_{1m}(Y_{1n})$ or simply $X(Y)$. $\Pr(\cdot|\cdot)$ represents a conditional probability. With these notations, the goal of interdependent decision problems is to predict

$$y^*_{1n} = \arg\max_{y_{1n}} \Pr(Y_{1n} = y_{1n}|X_{1m} = x_{1m}) \tag{5.1}$$

for each given input variables x_{1m}.

2.1 Modeling Full Conditional Probability

Arguably the most straight-forward approach is to directly model the conditional probability in Eq.(5.1). This is the approach we take for multi-topic text categorization in Chapter 3, where the scoring function of Eq.(3.5) corresponds to the conditional probability $\Pr(Y_{1n}|X_{1m})$, where X_i represents the i-th feature of text and Y_j the relevancy of the j-th topic.

Generally speaking, this approach is too naive since the number of possible output assignments can be so huge that a good estimation of $\Pr(Y_{1n}|X_{1m})$ or a corresponding scoring function can be quite hard due to data sparseness. For some situations, however, only small fraction of possible output assignments are actually observed and then a good modeling of the full conditional probability is possible. For example, in "Arts & Humanities" dataset of Table 3.2, there are 26 topics, thus $2^{26} \approx 10^8$ possible output assignments (i.e, labels). The number of observed assignments, however, is only 599 and much smaller than that of possible assignments. Although 599 classes are rather huge compared to usual multi-class categorization problems, the results of Chapter 3 show that multi-class categorization techniques can be successfully applied to such cases.

We think that the results of Chapter 3 suggest that the combination of powerful generalization techniques such as large margin principle and efficient implementation techniques can lead to good estimation of the full conditional probability when the number of observed output assignments are not so large.

2.2 Sequential Prediction

Using Bayes' rule, $\Pr(Y_{1n}|X_{1m})$ can be factorized into

$$\Pr(Y_{1n}|X_{1m}) = \prod_{k=1}^{n} \Pr(Y_{j_k}|Y_{j_1}, Y_{j_2}, \ldots, Y_{j_{k-1}}, X_{1m})$$

where $(j_1, j_2, \ldots, j_n)$ is a permutation of $(1, 2, \ldots, n)$. In some cases, we can assume conditionally independence properties and can reduce the number of conditional variables like

$$\Pr(Y_{j_k}|Y_{j_1}, Y_{j_2}, \ldots, Y_{j_{k-1}}, X_{1m}) = \Pr(Y_{j_k}|\mathcal{B}_{j_k}, X_{1m}), \tag{5.2}$$

where B_{j_k} is a (usually small) subset of $Y_{j_1}, Y_{j_2}, \ldots, Y_{j_{k-1}}$. From Eq.(5.1) and Eq.(5.2), we get

$$y^*_{1n} = \arg\max_{Y_{1n}} \prod_{k=1}^{n} \Pr(Y_{j_k}|\mathcal{B}_{j_k}, x_{1m}). \tag{5.3}$$

Now let's think the following assignment $\hat{y}_{j_k}$

$$\hat{y}_{j_k} = \arg\max_{Y_{j_k}} \Pr(Y_{j_k}|B_{j_k}, x_{1m}) \quad \text{for} \quad \forall k \in \{1, 2, \ldots, n\}, \tag{5.4}$$

where B_{j_k} represents the assignment of $\hat{y}_{j_1}, \hat{y}_{j_2}, \ldots, \hat{y}_{j_{k-1}}$ to the corresponding variables in $\mathcal{B}_{j_k}$. Although $\hat{y}_{1n}$ does not coincide with y^*_{1n} in general, $\hat{y}_{1n}$ can be good approximations of y^*_{1n} if the mass of $\Pr(Y_{j_k}|B_{j_k}, x_{1m})$ concentrates on a specific value of Y_{j_k} for all k. This condition can be stated more formally as the conditional entropy,

$$H(Y_{j_k}|B_{j_k}, x_{1m}) = -\sum_{Y_{j_k}} \Pr(Y_{j_k}|B_{j_k}, x_{1m}) \log \Pr(Y_{j_k}|B_{j_k}, x_{1m}), \tag{5.5}$$

should be small. If we have a good model of $H(Y_{j_k}|B_{j_k}, x_{1m})$, we can construct a deterministic approximation procedure for Eq.(5.3) as follows.

1. Find j_k which achieves the smallest value of Eq.(5.5).

2. Compute $\hat{y}_{j_k}$ using Eq.(5.4).

3. Repeat the above steps until all output variables are assigned.

Strategy learning approach in Chapter 4 can be seen as a discriminative modeling of the above approximation procedure: the strategy function $g(\mathbf{x}, \hat{\mathbf{y}})$ corresponds to the conditional entropy $H(Y_{j_k}|B_{j_k}, x_{1m})$ and the tag classifier $f(\mathbf{x}, \hat{\mathbf{y}})$ to the conditional probability $\Pr(Y_{j_k}|B_{j_k}, x_{1m})$. It is obvious that this correspondence can be established in general interdependent decision problems. Additionally the above discussion implies that strategy learning approach may be effective if $H(Y_{j_k}|B_{j_k}, x_{1m})$ is very small at least for one j_k.

3. Future Work

In this section, we list future work directions related to this thesis.

Classification-based Method for More General Selector In Section 4.1, we define two types of selector, n-best selector and p-percentile selector, which choose a set of items from input item sets. Although these selectors can represent a wide range of selection processes, this is not always the case in real selection problems.

For example, suppose that you are asked to select a representative sentence set from an article, say A, and select a sentence set S. Then suppose that the article is slightly updated and a new sentence t which summarizes well a part of S is inserted into the article. Now you can make more representative sentence set by removing a part of S which is summarized by t and adding t and other sentences to S. This situation cannot be expressed with n-best or p-percentile selector since if the selector (you) is a n-best or p-percentile selector, an addition of an item (t) must results in the replacement of a member of S with t, or change nothing.

The methods introduced in Chapter 2 cannot be applied directly to such situations. Thus it is an interesting question whether any classification-based method can be developed for more general selectors.

Kernel Design in Label Space We used Dice measure as a kernel for labels in Maximal Margin Labeling. An interesting question is how much effect the choice of kernel in label space has on MML's performance. We reported a preliminary

result on the effect of other label kernels in [44], but the result was far from conclusive.

Kernel design is one of the most important research topic in theory of kernel-based methods [25]. Most studies, however, concentrate on the design of kernel for inputs. Thus the design of kernel for *outputs* (labels) might be an interesting topics also from theoretical viewpoints.

Suggestion of New Topic When we investigated the experimental results in Chapter 3, we found some specific topic combinations are assigned to many texts, and thus behave like new topics. In such cases, it may be helpful to suggest users that these combinations might be considered new topics. Since it is not easy to tell whether such a combination really represents a new topic (like quantum computing) or just strongly-correlated topics for some reasons, this needs further studies.

Strategy Learning on More Complex Structure Recently, structured output problems have attracted many researchers' attention and are being studied intensively [45, 34, 35, 46, 47]. Most of such studies are based on the idea of solving structured output problems by learning a scoring function which evaluates the "goodness" of an input-output pair.

On the other hand, the strategy learning approach we proposed is based on quite a different idea of learning a strategy to predict structured outputs step-by-step. Thus it is a natural question whether this approach can be extended for more complicated structured outputs such as trees and graphs. Since such complex structures often appear in natural language processing, the question is not only interesting, but also important.

Proof of Convergence in Strategy Learning In the experiments in Chapter 4, we observed that the weight w of strategy function converged rather smoothly. However, we do not have any theoretical guarantee that this is always the case. From a practical point of view, it is important to know the conditions by which a convergence is guaranteed.

References

[1] E. Charniak. *Statistical Language Learning*. The MIT Press, 1996.

[2] I. Mani. *Automatic Summarization*. John Benjamins Publishing Company, 2001.

[3] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[4] N. Ueda and K. Saito. Parametoric mixture models for multi-topic text. *Advances in Neural Information Processing Systems*, 15:1261–1268, 2003.

[5] T. Hirao, H. Isozaki, E. Maeda, and Y. Matsumoto. Extracting important sentences with support vector machines. In *Proceedings of the Nineteenth International Conference on Computational Linguistics*, pages 342–348, 2002.

[6] K. Crammer and Y. Singer. Pranking with ranking. *Advances in Neural Information Processing Systems*, 14:641–647, 2002.

[7] J.A. Anderson and P.R. Philips. Regression, discrimination and measurement models for ordered categorical variables. *Applied Statistics*, 30(1):22–31, 1981.

[8] Y. Freund, R. Iyaer, R.E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 170–178, 1998.

[9] R. Herbrich, T. Graepel, and K. Obermayer. *Advances in Large Margin Classifiers*, chapter Large margin rank boundaries for ordinal regression, pages 115–132. The MIT Press, Cambridge, 2000.

[10] W.W. Cohen, R.E. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[11] K. Takeuchi. *Mathematical statistics*, pages 140–146. Toyo Keizai Shinpousha, Tokyo, 1963.

[12] H. Kazawa, T. Hirao, and E. Maeda. Order svm: a kernel method for order learning based on generalized order statistics. *Systems and Computers in Japan*, 36(1):35–43, 2005.

[13] V.N. Vapnik. *The Nature of Statistical Learning Theory, Second edition.* Springer-Verlag New York, Inc., 1999.

[14] J.T. Kwok. Moderating the outputs of support vector machine classifiers. *IEEE Transaction on Neural Networks*, 10(5):1018–1031, 1999.

[15] Thorsten Joachims. *Learning to Classify Text Using Support Vector Machines : Methods, Theory and Algorithms.* Springer, 2002.

[16] M. Nagata and H. Taira. Information-based inductions sciences : Text classification - showcase of learning theories -. *IPSJ Magazine*, 21(1):32–37, 2001.

[17] T. Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of the Tenth European Conference on Machine Learning*, pages 137–142, 1998.

[18] R. E. Schapire and Y. Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[19] Yoav Freund and Robert E. Schapire. A short introduction to boosting. *Jornal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.

[20] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[21] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001.

[22] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wealy, 1999.

[23] K. Morik, P. Brockhausen, and T. Joachims. Combining statistical learning with a knowledge-based approach - a case study in intensive care monitoring. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 268–277, 1999.

[24] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge Unversity Press, 2000.

[25] B. Schölkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, 2002.

[26] M. Iri. *Ippan Senkei Daisu (General Linear Algebra)*. Iwanami Shoten, 2003.

[27] A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142, 1996.

[28] T. Kudoh and Y. Matsumoto. Use of support vector learning for chunk identification. In *Proceedings of the Fourth Conference on Computational Language Learning*, pages 142–144, 2000.

[29] T. Kudo and Y. Matsumoto. Chunking with support vector machines. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.

[30] R. Koeling. Chunking with maximum entropy models. In *Proceedings of the Fourth Conference on Computational Language Learning*, pages 139–141, 2000.

[31] Y. Tsuruoka and J. Tsujii. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 467–474, 2005.

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

[33] E. Charniak, C. Hendrickson, N. Jacobson, and M. Perkowitz. Equations for part-of-speech tagging. In *National Conference on Artificial Intelligence*, pages 784–789, 1993.

[34] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 3–10, 2003.

[35] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. *Advances in Neural Information Processing Systems*, 16:25–32, 2004.

[36] E. F. T. K. Sang and S. Buchholz. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the Fourth Conference on Computational Language Learning*, pages 127–132, 2000.

[37] T. Zhang, F. Damerau, and D. Johnson. Text chunking using regularized winnow. In *Proceedings of the Thirty-Ninth Annual Meeting of the Association for Computational Linguistics and the Tenth Conference of the European Chapter of the Association for Computational Linguistics*, pages 539–546, 2001.

[38] X. Carreras and L. Màrquez. Phrase recognition by filtering and ranking with perceptrons. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing*, pages 205–216, 2003.

[39] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[40] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[41] M. Collins. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *Proceedings of the Fourteenth Annual Meeting of the Association for Computational Linguistics*, pages 489–496, 2002.

[42] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1–8, 2002.

[43] Y. Freund and R Schapire. Large margin classification. *Machine Learning using the Perceptron Algorithm*, 37(3):277–296, 1999.

[44] H. Kazawa, T. Izumitani, H. Taira, and E. Maeda. Maximal margin labeling for multi-topic text categorization. *Advances in Neural Information Processing Systems*, 17:649–656, 2005.

[45] J. Weston, O. Chapelle, A. Elisseeff B. Schölkopf, and V. Vapnik. Kernel dependency estimation. *Advances in Neural Information Processing Systems*, 15, 2002.

[46] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector learning for interdependent and structured output spaces. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 823–830, 2004.

[47] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 1–8, 2004.

Publication List

Journal Papers

1. Hideto Kazawa, Tsutomu Hirao, and Eisaku Maeda. Rank Estimation: A New Challenge to Machine Learning. *Information Technology Letters*, 1:113–114, 2002. (In Japanese)

2. Hideto Kazawa, Tsutomu Hirao, and Eisaku Maeda. Order SVM: A Kernel Method for Order Learning Based on Generalized Order Statistic. *The IEICE Transactions on Information and Systems, Pt.2 (Japanese Edition)*, J86-D-II(7):926–933, 2003. (In Japanese)

3. Tsutomu Hirao, Hideto Kazawa, Hideki Isozaki, Eisaku Maeda, and Yuji Matsumoto. Machine Learning Approach to Multi-document Summarization. *Journal of Natural Language Processing*, 10(1):81–108, 2003. (In Japanese)

4. Hideto Kazawa, Tomonori Izumitani, Hirtoshi Taira, Eisaku Maeda, and Hideki Isozaki. A Maximum Margin Learning of Multi-category Labeling. *The IEICE Transactions on Information and Systems, Pt.2 (Japanese Edition)*, J88-D-II(11):2246–2259, 2005. (In Japanese)

Journal Papers (Translation)

1. H. Kazawa, T. Hirao, and E. Maeda. Order SVM: A Kernel Method for Order Learning Based on Generalized Order Statistic. *Systems and Computers in Japan*, 36(1):35–43, 2005. (The original Japanese paper appeared in *The IEICE Transactions on Information and Systems, Pt.2 (Japanese Edition)*, J86-D-II(7):926–933, 2003.)

Conferences and Workshops (Refereed)

1. Hideto Kazawa, Kazunori Fujimoto, and Kazumitsu Matsuzawa. Attribute Dependency Acquisition From Formatted Text. In *Proceedings of Third International Conference on Knowledge-based Intelligent Information Engineering Systems (KES-99)*, pages 464–468, 1999.

2. Hideto Kazawa and Kazunori Fujimoto. Grading Properties According to the Locations of Their Descriptions in Press Releases. In *Proceedings of fourth Australian Knowledge Acquisition Workshop*, pages 30–43, 1999.

3. Hideto Kazawa and Eisaku Maeda. Learning Ordering Function from Partially Ranked Examples. In *Proceedings of Neural Information Processing Systems Workshop "Beyond Classification and Regression"*, 2002.

4. Hideto Kazawa, Tomonori Izumitani, Frédéric Tingaud, Hirotoshi Taira, and Eisaku Maeda. A Large Margin Method for Multi-labeling and Its Application to Gene Category Prediction. In *Proceedings of Neural Information Processing Systems Workshop "New Problems and Methods in Bioinformatics"*, 2003.

5. Hideto Kazawa, Tomonori Izumitani, Hirotoshi Taira, and Eisaku Maeda. Maximal Margin Labeling for Multi-topic Text Categorization. In *Advances in Neural Information Processing Systems*, volume 17, pages 649–656, 2005.

Conferences and Workshops (Not Refereed)

1. Hideto Kazawa, Tsutomu Hirao, and Eisaku Maeda. Ranking SVM and Its Application to Sentence Selection. In *Proceedings of Fifth Workshop on Information-Based Induction Sciences (IBIS-2002)*, pages 37–42, 2002. (In Japanese)

2. Hideto Kazawa, Tsutomu Hirao, and Eisaku Maeda. Learning Ordering Function from Ranked Examples. In *Techinical Report of IEICE*, volume PRMU2002-72/WIT2002-15, pages 11–16, 2002. (In Japanese)

3. Hideto Kazawa, Jun Suzuki, and Eisaku Maeda. SVMAP - A Large Margin Map Approximator. In *Proceedings of Sixth Workshop on Information-Based Induction Sciences*, pages 205–210, 2003. (In Japanese)

4. Hideto Kazawa, Tomonori Izumitani, Hirotoshi Taira, and Eisaku Maeda. Multi-topic Text Categorization to Maximize F-measure. In *Proceedings of the Tenth Annual Meeting of the Association for Natural Language Processing*, pages 689–692, 2004. (In Japanese)

5. Hideto Kazawa, Tomonori Izumitani, Hirotoshi Taira, and Eisaku Maeda. Maximum Margin Labeling for Multi-topic Text Categorization. In *IPSJ SIG Technical Report*, volume 2004-FI-76/2004-NL-163, pages 53–60, 2004. (In Japanese)

6. Toshihiro Kamishima, Hideto Kazawa, and Shotaro Akaho. Estimating Attributed Central Orders: An Empirical Comparison. In *Proceedings of the Fifteenth European Conference on Machine Learning and the Eighth European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD-2004)*, pages 563–565, 2004.

7. Toshihiro Kamishima, Hideto Kazawa, and Shotaro Akaho. Supervised Ordering – An Empirical Survey. In *Proceedings of Eighth Workshop on Information-Based Induction Sciences*, pages 213–218, 2005. (In Japanese)

8. Toshihiro Kamishima, Hideto Kazawa and Shotaro Akaho. Supervised Ordering: An Empirical Comparison. In *Proceedings of Fifth IEEE International Conference on Data Mining (ICDM-2005)*, pages 673–676, 2005.

Abbreviations

IPSJ Information Processing Society of Japan

IPSJ SIG Special Interest Group of IPSJ

IEICJ The Institute of Electronics, Information and Communication Engineers