Kernel Engineering on Parse Trees

SUN Jun

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in School of Computing

NATIONAL UNIVERSITY OF SINGAPORE

2011

Acknowledgments

I would like to express my sincere gratitude to my advisors, Professor Tan Chew Lim and Dr. Zhang Min for their guidance and support.

More than being the adviser on my research work, Prof. Tan also provides a lot of help on my life here in Singapore. Prof. Tan is always so considerate that he even cares us more than ourselves. With the support from Prof. Tan, I gain a lot of freedom in my research work so that I can have a chance to develop a broad background according to my interest.

As my co-supervisor, Dr. Zhang made a lot of effort in guiding my research capability from the scratch to being able to carry out research work independently. I feel so lucky to work with such an experienced and enthusiastic researcher. Every time I talked with him and asked for an advice, he could never ever let me down.

I would also like to thank my thesis examination committee including Prof. Ng Hwee Tou and Dr. Sim Khe Chai from the local university and Dr. Moschitti from University of Trento for their acceptance of reviewing my thesis and their valuable suggestions on refining this manuscript.

I specially dedicated this thesis to my family for their support over these years. Other than pursuing a Ph.D, almost whatever choices I have made during my life, they would always be there on my side as the strongest supporters. Words just amount to too little to express the full measure of my gratitude.

Finally I would like to thank everybody that has helped and inspired me esp. but not limited to people in CHIME lab (NUS) and members in MT group (I^2R).

List of Publications during Candidature

Sun J., M. Zhang, and C.L. Tan. 2011. Tree sequence kernel for natural language. In Proceedings of the 25th AAAI Conference on Artificial Intelligence. (AAAI-2011)

Sun J., M. Zhang, and C.L. Tan. 2010. Exploring syntactic structural features for sub-tree alignment using bilingual tree kernels. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 306-315. (ACL-2010)

Sun J., M. Zhang, and C.L. Tan. 2010. Discriminative induction of sub-tree alignment using limited labeled data. In Proceedings of the 23rd International Conference on Computational Linguistics, pages 1047-1055. (COLING-2010)

Sun J., M. Zhang, and C.L. Tan. 2009. A noncontiguous tree sequence alignment-based model for statistical machine translation. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, pages 914-922. (ACL-2009)

Chen, B., **Sun J.**, H. Jiang, M. Zhang and A.T. Aw. 2007. I2R Chinese-English Translation System for IWSLT 2007. In Proceeding of the 4th International Workshop on Spoken Language Translation. (IWSLT-2007)

Zhang, M., H. Jiang, A.T. Aw., **Sun J.**, S. Li and C.L. Tan. 2007. A tree-to-tree alignment-based model for statistical machine translation. In Proceeding of the Machine Translation Summit XI. (MT-summit-2007)

Contents

Abstract vii

List of Figures xi

List of Tables xiii

Chapter 1 Introduction 1

1.1 Aims and Contributions of the study 4

1.2 Outline of this Thesis 6

Chapter 2 Background 9

2.1 Kernel Methods 9

2.2 Support Vector Machines 18

2.3 Preliminary Concepts on Tree Structures 23

2.4 Summary 26

Chapter 3 Kernels on Discrete Structures 27

3.1 Representative Kernels 28

3.1.1 Haussler's Convolution Kernel 29

3.1.2 Mapping Kernel 30

3.1.3 Sequence Kernel 35

3.1.4 Collins and Duffy's Tree Kernel 37

3.2 Previous Work . . . 39
3.3 Summary . . . 43

Chapter 4 Tree Sequence based Kernels **45**

4.1 From Tree Kernel to Tree Sequence Kernel – Some Motivating Examples . . . 46
4.2 Contiguous Tree Sequence Kernel . . . 50
4.2.1 Kernel Evaluation via Pseudo Roots . . . 50
4.2.2 Algorithm 2: Fast Evaluation . . . 52
4.3 Set Sequence Kernels . . . 54
4.3.1 Penalizing Length of Spans (γ) . . . 57
4.3.2 Penalizing Count of Matched Elements (μ) . . . 58
4.3.3 Penalizing Count of Gaps (τ) . . . 59
4.4 Tree Sequence Kernels . . . 60
4.4.1 The Generalized Tree Sequence Kernel . . . 60
4.4.2 Adapting Set Sequence Kernel to Tree Sequence Kernel . . . 62
4.4.3 Penalizing Length of Spans (γ) . . . 64
4.4.4 Penalizing Count of Matched Subtrees (μ) . . . 65
4.4.5 Penalizing Count of Gaps (τ) . . . 65
4.5 Anchored Tree Sequence Kernel . . . 66
4.5.1 Feature Space Construction . . . 67
4.5.2 Penalizing Length of spans (γ) . . . 71
4.5.3 Penalizing Count of Matched Subtrees (μ) . . . 73
4.5.4 Penalizing Count of Gaps (τ) . . . 74
4.5.5 Mapping Kernels with Anchored Structures . . . 76
4.6 Kernels over Multiple Parse Trees . . . 79
4.6.1 Independent Bilingual Tree Kernel (iBTK) . . . 80
4.6.2 Dependent Bilingual Tree Kernel (dBTK) . . . 81

4.6.3 Generalized Kernels over Multiple Trees 83
4.7 Summary 84

Chapter 5 Tree Sequence Kernels for Single Parse Tree 85
5.1 Background and Related Work 86
5.2 Experiments 90
5.2.1 Question Classification 91
5.2.2 Relation Extraction 94
5.3 Discussion 101
5.4 Summary 102

Chapter 6 A Kernel based Statistical Model for Bilingual Subtree Alignment 105
6.1 Task Definition 106
6.2 Background and Related Work 107
6.3 Structure Space for BT(S)Ks 109
6.4 Heuristic Feature Functions 113
6.4.1 Lexical and Word Alignment Features 113
6.4.2 Online Structural Features 115
6.5 The Alignment Model 115
6.6 Experiments on Bilingual Subtree Alignment 116
6.6.1 Data Preparation 117
6.6.2 Baseline approaches 119
6.6.3 Experimental Settings 120
6.6.4 Experimental Results 121
6.7 Experiments on Machine Translation 125
6.7.1 Experimental Settings 125
6.7.2 Experimental Results 126

6.8 Summary 129

Chapter 7 Conclusion **131**

7.1 Summary of Achievements 131

7.2 Future Directions 134

Reference **135**

Abstract

Recently, Natural Language Processing (NLP) has been greatly benefiting from the progress of machine learning methods in large data driven applications. Some NLP tasks require complex data representation to deeply analyze the syntactic and semantic features. In many cases the input data is represented as sequences, trees and even graphs. Traditional feature based methods transform these structured input data into vectorial representation by sophisticated feature engineering, which is argued infeasible to fully explore the structure features. Alternatively, kernel methods can explore a very high dimensional feature space for these complex input structures without explicitly representing the input data as a feature vector. In terms of tree structures, tree kernels can explore the subtree features in the parse trees, without explicitly enumerating each type of subtree.

However, previous tree kernels explore the structure features with respect to the single subtree representation. The structure of the large single subtree may be sparse in the data set, which prevents large structures from being effectively utilized. Sometimes, only certain parts of a large subtree are beneficial instead of the entire subtree. In this case, using the entire structure may introduce noisy information.

To address the above deficiency, this dissertation systematically investigates the phrase parse tree and attempts to design more sophisticated kernels to deeply explore the structure features embedded in the phrase parse trees other than the single subtree representation.

In order to achieve this goal, this dissertation proposes *tree sequence* based kernels to explore the structure features in the phrase parse trees for various NLP applications.

Specifically, this thesis achieves to propose contiguous Tree Sequence Kernel (cTSK) which is able to capture the structure features of a contiguous subtree sequence, and to propose an efficient algorithm to evaluate the kernel function. In addition, this thesis proposes Tree Sequence Kernels (TSKs) which are able to capture the structure features of a subtree sequence, both contiguous and non-contiguous. Particularly, the generalized TSK is verified to be a valid positive semidefinite kernel by being constructed on the mapping kernel paradigm. Additionally, the generalized TSK is instantiated by weighting the structure features through a variety of schemes. Efficient algorithms are proposed to evaluate the kernel functions based on these different structure weighting schemes. Based on TSKs, this thesis proposes Anchored Tree Sequence Kernels (aTSKs) to match the subtree sequence structures according to their relative spatial relation with certain anchored structures. From this perspective, aTSKs are able to facilitate the applications, which deal with the instances with multiple relational target constituents, i.e. relation extraction.

To test the effectiveness of the proposed tree sequence based kernels, this thesis applies the proposed kernels to two NLP applications, i.e. Question Classification and Relation Extraction, and conducts comparative experiments to demonstrate the characteristics of the proposed kernels. Experimental results show that the tree sequence based kernels outperform the tree based kernels on both tasks, which suggests that the structure features expressed as a sequence of subtrees are very effective for those tasks.

This thesis additionally extends the tree (sequence) based kernels from the single parse tree to multiple parse trees. This is achieved by exploiting the structure features across multiple trees both dependently and independently. The proposed kernels on multiple parse trees are instantiated on the bilingual task, i.e. Bilingual Subtree Alignment. Experimental results suggest that the corresponding Bilingual

Tree (Sequence) Kernels can effectively explore the structure features for this task.

Another contribution of this thesis is to propose a generic framework for the task of Bilingual Subtree Alignment by means of a kernel based statistical model. In addition, this thesis integrates the tree (sequence) based kernels on bilingual parallel parse trees in this framework along with various heuristic feature functions. Experimental results reveal that the aligned subtrees obtained from the proposed framework can be well utilized as translation rules by a variety of the state-of-the-art statistical machine translation systems.

In all, this dissertation adopts the subtree sequence structure as the basic feature type for kernels on phrase parse tree. A variety of kernels are built up based on the subtree sequence structure. The advantages of the subtree sequence structures are demonstrated on various NLP applications. By means of the tree (sequence) kernels over multiple parse trees, a kernel based alignment model is proposed for the task of bilingual subtree alignment, with which the translation performance can be effectively improved. On a more general perspective, this dissertation systematically explores the disconnected structure features in parse trees by means of kernels. On this point, this dissertation may provide novel views of structure features for NLP applications.

List of Figures

2.1 The linearly separable case for SVM 19
2.2 The linearly nonseparable case for SVM 22
2.3 Tree Structure Illustration 25

3.1 Collins and Duffy's Tree Kernel feature space. 37
3.2 Illustration of feature space of Partial Tree Kernel (PTK) 40

4.1 Illustration of Tree Sequence Structures. 47
4.2 Illustration of Pseudo Root construction 51
4.3 Construction of Node Set Sequence 63
4.4 Tree Structure Partition. 69

5.1 Parse tree instance for relation extraction 94
5.2 Illustration of Tree Sequence Structures. 95
5.3 Non-contiguous tree sequence features 101

6.1 Subtree alignment as referred to Node alignment 107
6.2 Illustration of SST, RdSST and RgSST 111
6.3 Alignment results comparison 128

List of Tables

5.1 Accuracy of Question Classification 92
5.2 Performance for Relation Extraction on MCT 97
5.3 Performance for Relation Extraction on PT 97
5.4 Constraint PTKs on MCT 100
5.5 Constraint PTKs on PT 100

6.1 Corpus Statistics for HIT corpus 118
6.2 Statistics of FBIS selected Corpus 118
6.3 Structure feature contribution for HIT test set 121
6.4 Structure feature contribution for FBIS test set 122
6.5 MT evaluation on various systems 127

Chapter 1

Introduction

Recently, enormous applications on the Internet have been developed, which are extensively guiding and shaping people's life. Among them, the text based applications are particularly popular such as Search Engines, Blogs, Micro-Blogs as well as other web social mining applications. Along with the development of these text based web applications, language processing techniques have become more and more important, since they can help these web applications provide better user experience with more accurate results. This requirement strongly stimulates the research in Natural Language Processing (NLP) which has made quick progress in developing data driven applications. The NLP research domain also benefits from the availability of a large amount of natural language data and the improvement of the large scale statistical machine learning methods.

In the field of NLP, the most familiar data representation is the sequence (string), since a document on the web can be considered as a sequence of words or a sequence of alphabets. There are a lot of applications that are built on the sequence representation. In document classification, a given document represented as a sequence of words is classified into categories such as "Politics", "Sports", "Business", etc. In sentiment analysis, the sentiment preference (like or dislike,

recommend or not recommend) is required to be detected for a given text which is also represented as a sequence of words.

These applications actually rely on many fundamental text analysis techniques which have been developed for the sequential data. In addition, these techniques can be built on more complex data representations. Part-of-speech tagging annotates a tag to each word in a sentence, which denotes its grammatical function in the sentence. In this application, the original sequence of words is attached in parallel with the other sequence of part-of-speech tags. Syntactic parsing creates a tree structure over a sentence with each of the node in the tree to denote the grammatical function of the span of text it covers. Thus, the sequence of words is enriched with a tree structure as the grammatical analysis. By means of these augmentations of the input data, the input data can be represented by more complex structures such as multiple sequences, trees and even graphs other than a sequence of plain words.

Traditionally, statistical machine learning methods adopt the vectorial representation to express the features embedded in those complex structures. Generally, a preprocessing step is required to transform those structured data into the corresponding feature vector. Then the obtained feature vector is used as the input of certain machine learning algorithms employed to solve the task. Since this preprocessing step is task related, it often requires an expert to appropriately design and extract the features, which is obviously non-trivial. In addition, this feature engineering process may lead to the loss of information of the input data.

To better perform these structural data driven tasks, kernel methods are introduced, which can directly take these structures as the input for kernel machines[1] instead of explicitly representing the input data as a feature vector. By appropriately designing a kernel function, high dimensional feature space can be implicitly explored. Thus, the similarity between two data objects represented by the dot

[1] A kernel machine refers to a machine learning algorithm built on kernel methods

product of the high dimensional feature vectors can be efficiently evaluated. Kernel methods have been integrated into many learning machines, such as Perceptron (Rosenblatt, 1962) and Support Vector Machines (SVM) (Cortes and Vapnik, 1995).

As previously introduced, the input data structures in NLP applications often consist of sequences, trees and graphs. Consequently, kernel methods can be applied in these structures, i.e. sequence kernel (Lodhi et al., 2002), tree kernel (Collins and Duffy, 2002) and graph kernel (Suzuki, Sasaki, and Maeda, 2006). Among these, tree kernel (Collins and Duffy, 2002) explores the syntactic substructure space of all subtree types. To evaluate the kernel function means to count the number of common subtrees in each type as the similarity between two syntactic parse trees. Tree kernel has been successfully applied in many NLP tasks such as syntactic parsing (Collins and Duffy, 2002), question classification (Zhang and Lee, 2003; Moschitti, 2006), semantic parsing (Moschitti, 2004; Zhang et al., 2008a), textual entailment (Zanzotto and Moschitti, 2006), relation extraction (Zelenko, Aone, and Richardella, 2003; Zhang, Zhou, and Aw, 2008) as well as pronoun resolution (Yang, Su, and Tan, 2006).

Based on the standard Collins and Duffy's tree kernel which explores all the subtree structures, recent work attempts to deeply explore the substructure features beyond the subtree structure in the original parse tree. Partial tree kernel (Moschitti, 2006) allows partial production rule matching within a syntactic structure, which extensively enlarges the feature space. However, the partial matching between subtrees also introduces many non-grammatical substructures which violate the original production rules (Moschitti, 2006). In order to extend the feature space with more meaningful substructures, grammar driven tree kernel (Zhang et al., 2008a) constructs a larger substructure feature space by modifying the original subtrees and generalizing certain grammar tags. However, the modification of

the subtrees is based on certain grammars, which requires human effort to create a set of beneficial grammar representations and is difficult to be extended to other languages and grammars.

Basically, these works differ in the subtree feature space explored by the kernels. However, it is worthwhile to point out that most of these studies explore the structure features with respect to the single subtree based features. Therefore, it is very likely that when covering a large context, the structure of a single subtree may be fairly large so that the data sparseness problem may prevent large structures from being effectively utilized (Collins and Duffy, 2002). In addition, it is also possible that a large single subtree is not entirely beneficial, but only certain parts are useful for the task. Therefore, using the entire subtree may introduce noisy information.

1.1 Aims and Contributions of the study

To address the deficiency of the single subtree structures, the main aim of this study is to investigate the phrase parse tree structure and attempt to design more sophisticated kernels to deeply explore the substructures embedded in the phrase parse trees other than the single subtree representation.

The proposed kernels are expected to preserve the production rule as it is in the original parse tree, since the production rule is the minimum informative structure to express the grammatical function of the span it covers. The proposed kernels should explore the feature space other than the single subtree structures. Specifically, the feature space is extended from the structure of a single subtree to the structure of multiple subtrees. The structure of multiple subtrees can be either a sequence of subtrees from one parse tree or multiple subtree sequences from multiple parse trees. In addition, the proposed kernels should be grammar independent and language independent, which allows the kernels to be easily extended to various

NLP applications across different languages.

The specific aims and contributions of this research are:

- to propose Tree Sequence Kernels (TSKs) to explore the structure space of a sequence of subtrees, both contiguous and non-contiguous. When the subtree sequence is restricted to be contiguous, TSKs[2] are simplified into a special case, namely contiguous Tree Sequence Kernel (cTSK). TSKs can effectively capture the syntactic features over a large context or a non-contiguous context, since TSKs enlarge the feature space by capturing not only the connected substructures of a single subtree, but also the disconnected substructures of a sequence of subtrees. cTSK/TSKs can effectively utilize the large structures by decomposing and matching them in parts, which may alleviate the sparseness of large structures. (Sun, Zhang, and Tan, 2011)

- to propose Anchored Tree Sequence Kernel (aTSKs) to further adapt TSKs to tasks with relational target constituents in the parse trees. Constructed under the paradigm of Mapping kernel (Shin and Kuboyama, 2008), aTSKs are able to preserve the position of the subtree sequence structures relative to the anchored structures that cover the target constituents. This makes the structure features more differentiable, especially for tasks modeling the relationship for target constituents, such as relation extraction.

- to propose a series of efficient algorithms to adapt the generalized form of TSKs/aTSK to different realization of weighting the substructures. The substructures can be weighted by the span length covered by the subtree sequence. In addition, the substructures can be weighted by the number of subtrees or the number of gaps in the subtree sequence.

[2]We refer TSKs to the kernels which allow both contiguous and non-contiguous subtree sequence structures.

- to extend the kernels from the single parse tree to multiple parse trees. The substructures across multiple parse trees are explored both dependently and independently. These extensions can be applied with the tree (sequence) based kernels, i.e. tree kernel, cTSK, TSKs, aTSKs, to tasks requiring multiple parse trees, such as multilingual applications using parallel parse trees on multilingual text. (Sun, Zhang, and Tan, 2010b)

- to propose a kernel based statistical alignment model for the task of Bilingual Subtree Alignment. Tree (sequence) based kernels on bilingual parse trees are integrated in this framework to explore the structure features along with a variety of heuristic features. The aligned subtrees are adopted as translation rules which are verified to be able to improve both phrase and syntax based statistical machine translation systems (Sun, Zhang, and Tan, 2010b; Sun, Zhang, and Tan, 2010a)

The kernels proposed in this study leverage on the ability of sequence kernel to capture the horizontal sequence structure and the ability of the single subtree based kernel to capture the vertical parse tree structure. The tree sequence structure explored by the proposed kernels is motivated by the decent effectiveness of non-syntactic phrases in tree sequence based syntax translation (Zhang et al., 2008b; Sun, Zhang, and Tan, 2009), which suggests that the disconnected structures beyond the syntactic constraint are very useful in translational equivalence modeling. Compared with the single tree based kernels, the additional subtree sequence structures enhance the modeling of the large structures and discrete structures. The proposed kernels tend to bring novel views of structure features in NLP.

1.2 Outline of this Thesis

This thesis is organized as follows:

In Chapter 2, the background and preliminary knowledge of this thesis are introduced. Basically, a brief introduction of kernel methods and Support Vector Machines (SVM) is given in this chapter. Two prevalent definitions of kernels are presented and are unified by being proved equivalent in this chapter. In addition, some preliminary concepts and definitions on tree structures which will appear later in this thesis are presented.

In Chapter 3, some literature studying kernels on discrete structures are reviewed. Two approaches to construct kernels on discrete structures are introduced, i.e. the convolution kernel paradigm (Haussler, 1999) and the mapping kernel paradigm (Shin and Kuboyama, 2008). The sequence kernel (Lodhi et al., 2002) and the tree kernel (Collins and Duffy, 2002) are also introduced since they serve as the basis for the construction of tree sequence based kernels in the following chapters.

In Chapter 4, a series of tree sequence based kernels are proposed. The elaboration starts with the most simple case, i.e. contiguous Tree Sequence Kernel (cTSK), which explores the feature space of a contiguous subtree sequence. cTSK is then generalized to the more general case, i.e. Tree Sequence Kernels (TSKs), which allows the subtree sequence to be non-contiguous. In order to efficiently evaluate TSKs, Set Sequence Kernel (SSKs) are proposed to evaluate the similarity between the sequence of node sets. Three weighting schemes are proposed for the generalized SSK and TSK. In addition, we propose the anchored Tree Sequence Kernel (aTSKs) to accommodate the anchored substructures in the parse trees. The chapter is concluded after showing that those proposed kernels can be further extended from the single parse tree to multiple parse trees, which are expected to facilitate multilingual applications in NLP.

In Chapter 5, the proposed tree sequence based kernels are applied on two monolingual NLP tasks, i.e. question classification and relation extraction, since

kernel methods on these tasks are well studied and are shown to achieve the state-of-the-art performance against the feature based methods. Specially, cTSK and TSKs are applied in both tasks while aTSK is only applied in relation extraction due to its characteristics in modeling the entity pairs, which can be considered as the anchored structures in the parse trees.

In Chapter 6, the tree (sequence) based kernels are applied in a bilingual task, i.e. Bilingual Subtree Alignment, to explore the structure features in the bilingual parse tree pair. A statistical subtree alignment model is proposed to integrate the bilingual tree (sequence) kernels. Besides the tree (sequence) based kernels to explore the structure features, some heuristic feature functions are proposed to explore the lexical features as well. The aligned subtrees obtained from the subtree aligner are then used as additional translation rules in various statistical machine translation systems. Evaluations are carried out in both subtree alignment and machine translation.

In Chapter 7, this thesis is concluded with the major accomplishments and findings. Some research directions are discussed to facilitate the research work in the future.

Chapter 2

Background

In this chapter, some background knowledge of this thesis will be given. Specially, the basis of kernel methods will be presented. Two prevalent definitions of the positive semidefinite kernel will be discussed and proved to be equivalent. The widely used kernel machine, i.e. Support Vector Machines (SVM), will be introduced, since the proposed kernels in this thesis will be applied by means of SVM. At last, some preliminary concepts on tree structures will be given to facilitate the further understanding of this thesis.

2.1 Kernel Methods

In pattern recognition, patterns are learned from the training data and are expected to generalize to the unseen data. A direct approach of achieving this is to compare the similarity between an unseen instance and the annotated instances in the training data. The similarity between data instances can be evaluated as:

$$\begin{aligned} K &: \mathcal{X} \times \mathcal{X} \to \mathbb{R} \\ Sim(x, x') &= K(x, x') \end{aligned} \tag{2.1}$$

where given two data points x and x', the similarity between them is a real number denoted as $K(x, x')$. In addition, K can be assumed to be symmetric, i.e. $K(x, x') = K(x', x)$ for all x and x'.

To evaluate the function K, it is necessary to design a similarity measurement between data points. If all data points $x \in \mathcal{X}$ can be represented as an N-dimensional feature vector $\vec{x} \in \mathbb{R}^N$, with each component to be x_i, $1 \leq i \leq N$. A simple method of similarity computation is to use the dot product between the two vectors $\vec{x}, \vec{x}' \in \mathbb{R}^N$, which can be defined as follows:

$$K(x, x') = \langle \vec{x}, \vec{x}' \rangle = \sum_{i=1}^{N} x_i \cdot x'_i \tag{2.2}$$

In the Euclidean space, dot product between two normalized vectors $\vec{x}$ and $\vec{x}'$ can be geometrically interpreted as the cosine of the angle θ between the vectors.

$$\cos(\theta) = \frac{\langle \vec{x}, \vec{x}' \rangle}{\|\vec{x}\| \cdot \|\vec{x}'\|} \tag{2.3}$$

To measure the similarity between data points using the dot product, it requires the data point to be represented as a vector. However, in real life applications, data points may be complex structures such as sequences, trees, graphs etc., which are not explicitly expressed as a feature vector. As a result, these input data with non-vectorial representation are required to be projected to a dot product space $\mathcal{H}$ with M dimensions. In some cases, even the input data is represented as a vector, a more expressive vectorial representation is still needed to encode more useful information of the input data. Both cases can be dealt with by creating a mapping function $\mathbf{\Phi} : \mathcal{X} \to \mathcal{H}$, satisfying

$$\mathbf{\Phi}(x) = (\Phi_1(x), \Phi_2(x), \ldots, \Phi_M(x))^T \tag{2.4}$$

By projecting the data point into a dot product space $\mathcal{H}$, the similarity function can be computed as follows:

$$K(x, x') = \langle \mathbf{\Phi}(x), \mathbf{\Phi}(x') \rangle = \sum_{m=1}^{M} \Phi_m(x) \Phi_m(x') \tag{2.5}$$

By far, we have introduced a symmetric function 2.5 to measure the similarity between data points using dot product. The dot product can be evaluated by projecting the original data point to a new space. This indicates that to evaluate this kind of similarity, it requires to design a mapping function to create the image of the data point in the dot product space. The following example is given to elaborate how such mapping can be achieved.

Given the data point x represented by $\vec{x} = (x_1, x_2)^T$ in the two dimensional space $\mathbb{R}^2$, the mapping function $\mathbf{\Phi}$ is able to project $\vec{x}$ in $\mathcal{X} = \mathbb{R}^2$ to an image in $\mathcal{H} = \mathbb{R}^6$ as follows:

$$\begin{aligned} \mathbf{\Phi} : \mathcal{X} = \mathbb{R}^2 &\rightarrow \mathcal{H} = \mathbb{R}^6 \\ (x_1, x_2)^T &\rightarrow (1, x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2)^T \end{aligned} \tag{2.6}$$

For this case, it is convenient to list all the components in the projected vector, since it is only with six dimensions. However, when the dimension of the projected space is high or even infinite, explicitly enumerating the features will be infeasible. Therefore, in order to project the data point to the high dimensional dot product space, it is better to avoid the explicit mapping. In fact, the dot product between the projected data images can be evaluated by means of the dot product of the data points in the input space. For the previous example, given $x, x' \in \mathbb{R}^2$,

$$\begin{aligned} K(x, x') &= \langle \mathbf{\Phi}(\vec{x}), \mathbf{\Phi}(\vec{x}') \rangle \\ &= 1 + x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2(x_1x_2)(x_1'x_2') + 2x_1x_1' + 2x_2x_2' \\ &= (1 + \langle \vec{x}, \vec{x}' \rangle)^2 \end{aligned} \tag{2.7}$$

As shown in Eq. 2.7, the kernel function K can be evaluated by the square of the dot product of the vectors in the original space $\mathcal{X}$.

Given this example, it is easy to see how the similarity function $K(x, x')$ can be implicitly evaluated. This is equivalent to the explicit evaluation by first mapping the data into the dot product space and then evaluating the dot product between the

projected data images. In fact, this similarity is the so called kernel function. Based on the above introduction, we give the definition of kernel presented in Shawe-Taylor and Cristianini (2004) and Herbrich (2002).

Definition 2.1 [Kernel (Shawe-Taylor and Cristianini, 2004; Herbrich, 2002)] *The function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel for all $x, x' \in \mathcal{X}$ if and only if*
(1) *There exists a mapping from the input space to the dot product space, i.e.*

$$\mathbf{\Phi} : \mathcal{X} \to \mathcal{H} \tag{2.8}$$

(2) *K evaluates the dot product of the projected data image in $\mathcal{H}$, i.e.*

$$K(x, x') = \langle \mathbf{\Phi}(x), \mathbf{\Phi}(x') \rangle \tag{2.9}$$

Based on this definition, the input data points evaluated by the kernel function can be represented beyond the vectors and can be composed of complex structures. This definition directly corresponds to the process of the construction of a kernel function. In addition, some literature offer an alternative definition of kernel function (Schölkopf and Smola, 2002).

Definition 2.2 [(Positive Semidefinite) Kernel (Schölkopf and Smola, 2002)] *The function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel for any N data points $x^1, x^2, \ldots, x^N \in \mathcal{X}$ if and only if*
(1) *K is symmetric, i.e.*

$$K(x^i, x^j) = K(x^j, x^i) \tag{2.10}$$

(2) *the matrix, namely Gram Matrix, defined by $\mathbf{K}_{ij} = K(x^i, x^j)$ is positive semidefinite, i.e.*

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j \mathbf{K}_{ij} \geq 0 \tag{2.11}$$

for all $c_1, c_2, \ldots, c_N \in \mathbb{R}$

It can be found that Definition 2.1 interprets kernels for individual paired data points, while Definition 2.2 focuses on the characteristics demonstrated by a set of data points, i.e. data points are paired to form a Gram Matrix. In fact, the two definitions are equivalent to each other. When one serves as the definition, the other can be considered as the necessary and sufficient condition for K to be a valid kernel. To prove the equivalence of the two definitions, we will need

Lemma 2.1 [Cauchy-Schwarz Inequality] *If K is a kernel based on Definition 2.2, for all $x, x' \in \mathcal{X}$, there exists*

$$|K(x, x')|^2 \leq K(x, x) \cdot K(x', x') \tag{2.12}$$

Proof. See Schölkopf and Smola (2002).

Next we present the proof of the equivalence of the two definitions.

Theorem 2.2 [Equivalence of Kernel Definitions] *Definition 2.1 and Definition 2.2 are equivalent.*

Proof. Def. 1 $\Rightarrow$ Def. 2

First, we prove that given that kernel is defined in Definition 2.1, the two conditions presented in Definition 2.2 are satisfied.

Let $\mathbf{\Phi} : \mathcal{X} \to \mathcal{H}$ be a mapping function, such that for all $x^i \in \mathcal{X}$, $\mathbf{\Phi}(x^i) = \{\Phi_m(x^i)\}$, where $1 \leq m \leq M$. For any N data points $x^1, x^2, \ldots, x^N \in \mathcal{X}$, kernel is a function, $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, i.e. $K(x^i, x^j) = \langle \mathbf{\Phi}(x^i), \mathbf{\Phi}(x^j) \rangle$. Def. 2.1 defines K as a dot product, which is symmetric by definition. Alternatively, we have

$$\begin{aligned} K(x^i, x^j) &= \langle \mathbf{\Phi}(x^i), \mathbf{\Phi}(x^j) \rangle \\ &= \sum_{m=1}^{M} \Phi_m(x^i) \Phi_m(x^j) \\ &= \sum_{m=1}^{M} \Phi_m(x^j) \Phi_m(x^i) \\ &= \langle \mathbf{\Phi}(x^j), \mathbf{\Phi}(x^i) \rangle = K(x^j, x^i) \end{aligned}$$

which also shows that K is symmetric. In addition, for all $c_1, c_2, \ldots, c_N \in \mathbb{R}$, there exists

$$\begin{aligned}\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j K(x^i, x^j) &= \sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j \langle \mathbf{\Phi}(x^i), \mathbf{\Phi}(x^j)\rangle \\ &= \langle \sum_{i=1}^{N} c_i \mathbf{\Phi}(x^i), \sum_{j=1}^{N} c_j \mathbf{\Phi}(x^j)\rangle \\ &= \langle \sum_{i=1}^{N} c_i \mathbf{\Phi}(x^i), \sum_{i=1}^{N} c_i \mathbf{\Phi}(x^i)\rangle \geq 0\end{aligned}$$

which proves that the Gram Matrix defined by $\mathbf{K}_{ij} = K(x^i, x^j)$ is positive semidefinite. Hence, Definition 2.1 implies the properties defined in Definition 2.2.

Def.2 $\Rightarrow$ Def.1

Second, we will prove provided that kernel is defined by Definition 2.2, the two conditions presented in Definition 2.1 are satisfied. To achieve this,

- we first construct a mapping function, which maps the input data point into a new space.
- we then need to define the dot product on the new space so that the dot product between the projected images equals to the kernel of the input data points.

Suppose K is a kernel which satisfies the conditions in Definition 2.2. We define a map from $\mathcal{X}$ to a function space $\mathcal{H} = \{f : \mathcal{X} \to \mathbb{R}\}$, i.e.

$$\begin{aligned}\mathbf{\Phi} : \mathcal{X} &\to \mathcal{H} \\ x &\to K(\cdot, x)\end{aligned} \tag{2.13}$$

where given an input point x, the map will return a function $K(\cdot, x)$. The first argument of the kernel is the variable of the function.

Thus, the next step is to define a dot product $\langle \cdot, \cdot \rangle$ in $\mathcal{H}$ by means of the two conditions in Definition 2.1 and then to prove that

$$K(x, x') = \langle \mathbf{\Phi}(x), \mathbf{\Phi}(x') \rangle \tag{2.14}$$

However, instead of directly endowing a dot product in $\mathcal{H}$, we first turn $\mathcal{H}$ into a linear space $\mathcal{H}_0$, where

$$\mathcal{H}_0 = \left\{ f | f(\cdot) = \sum_{i=1}^{n} \alpha_i K(\cdot, x^i), \alpha_i \in \mathbb{R}, x^i \in \mathcal{X}, 1 \leq i \leq n \right\} \tag{2.15}$$

and then endow a dot product $\langle \cdot, \cdot \rangle$ in $\mathcal{H}_0$. In Eq. 2.15, when we let $\alpha_i = 1$ and $n = 1$, we will have $f(\cdot) = K(\cdot, x^1)$. Since $x^1, x^2, \ldots, x^n \in \mathcal{X}$ are arbitrary, all functions in $\mathcal{H}$ are included in $\mathcal{H}_0$. In fact, the space $\mathcal{H}_0$ is comprised of all the linear combinations of functions in $\mathcal{H}$. Consequently, a well-defined dot product in $\mathcal{H}_0$ is also valid for space $\mathcal{H}$.

Next we define a dot product in the space $\mathcal{H}_0$. Given $f, g \in \mathcal{H}_0$, i.e. $f(\cdot) = \sum_{i=1}^{n} \alpha_i K(\cdot, x^i)$ and $g(\cdot) = \sum_{j=1}^{m} \beta_j K(\cdot, x^j)$, where $\alpha_i, \beta_j \in \mathbb{R}$, $x^i, x^j \in \mathcal{X}$ for $1 \leq i \leq n$ and $1 \leq j \leq m$, the dot product is defined as

$$\langle f, g \rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j K(x^i, x^j) \tag{2.16a}$$

$$= \sum_{j=1}^{m} \beta_j f(x^j) \tag{2.16b}$$

$$= \sum_{i=1}^{n} \alpha_i g(x^i) \tag{2.16c}$$

Note that Eq. 2.16b suggests that the dot product does not depend on x^i and α_i, while Eq. 2.16c suggests that it does not depend on x^j and β_j. As a result, the defined dot product does not depend on the way of choosing the parameters and base kernel functions. In other words, the defined dot product is a property of the space $\mathcal{H}_0$ rather than a particular expansion of the functions f and g.

In addition, it can be proved that $\langle \cdot, \cdot \rangle$ obtains the following properties.

$$\langle f, g \rangle = \langle g, f \rangle \text{ for } f, g \in \mathcal{H}_0 \tag{2.17}$$

$$\langle cf + g, h \rangle = c\langle f, h \rangle + \langle g, h \rangle \text{ for } f, g, h \in \mathcal{H}_0 \text{ and } c \in \mathbb{R} \tag{2.18}$$

$$\langle f, f \rangle = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j K(x^i, x^j) \geq 0 \tag{2.19}$$

$$\langle f, f \rangle = 0 \text{ only if for all } x \in \mathcal{X},\ f(x) = 0 \tag{2.20}$$

From Eq. 2.16, it is seen to be proved that $\langle \cdot, \cdot \rangle$ is symmetric (Eq. 2.17) and bilinear (Eq. 2.18). In addition, Inequality 2.19 holds, since K is a positive definite kernel.

Next, we **prove the property 2.20**.

Before proving this property, we show that $\langle \cdot, \cdot \rangle$ is actually a valid kernel. Since we define kernel based on Definition 2.2 and the symmetric property satisfies as shown in Eq. 2.17. We need to prove the Gram Matrix defined for this kernel is positive semidefinite. Therefore, given $\gamma_1, \gamma_2, \ldots, \gamma_n \in \mathbb{R}$ and $f_1, f_2, \ldots, f_N \in \mathcal{H}_0$, we have

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \gamma_i \gamma_j \langle f_i, f_j \rangle = \langle \sum_{i=1}^{N} \gamma_i f_i, \sum_{j=1}^{N} \gamma_j f_j \rangle \tag{2.21a}$$

$$= \langle \sum_{i=1}^{N} \gamma_i f_i, \sum_{i=1}^{N} \gamma_i f_i \rangle \tag{2.21b}$$

$$= \langle f, f \rangle \geq 0 \tag{2.21c}$$

where Eq. 2.21a satisfies due to the bilinear property of $\langle \cdot, \cdot \rangle$ (Property 2.18), while Inequality 2.21c satisfies due to Property 2.19. As a result, $\langle \cdot, \cdot \rangle$ is a valid kernel based on Definition 2.2.

Based on Eq. 2.16, we have additional properties, namely reproducing proper-

ties,

$$\langle f, K(\cdot, x)\rangle = f(x) \tag{2.22a}$$

$$\langle K(\cdot, x), K(\cdot, x')\rangle = K(x, x') \tag{2.22b}$$

Then, based on Property 2.22 and Lemma 2.1 which is applied under the just proved condition that $\langle \cdot, \cdot \rangle$ is a valid kernel, we have

$$|f(x)|^2 = |\langle f, K(\cdot, x)\rangle|^2 \leq K(x, x) \cdot \langle f, f\rangle \tag{2.23}$$

Therefore, $\langle f, f\rangle = 0$ if and only if for all $x \in \mathcal{X}$, $f(x) = 0$. Hence, **Eq. 2.20 holds.**

Given that $\langle \cdot, \cdot \rangle$ obtains the properties of 2.17,2.18,2.19 and 2.20, we can claim that $\langle \cdot, \cdot \rangle$ is a well-defined dot product. In consequence, $\mathcal{H}_0$ is a well-defined dot product space.

By far, we have achieved constructing a well-defined dot product space $\mathcal{H}_0$ and obtaining the reproducing property 2.22 of the kernel K. Nevertheless, the original goal is to prove Eq. 2.14 in the space $\mathcal{H}$. Thus, for all $K(\cdot, x), K(\cdot, x') \in \mathcal{H}$, the condition $\mathcal{H} \subseteq \mathcal{H}_0$ implies $K(\cdot, x), K(\cdot, x') \in \mathcal{H}_0$. Based on the reproducing properties,

$$\langle \mathbf{\Phi}(x), \mathbf{\Phi}(x')\rangle = \langle K(\cdot, x) \cdot K(\cdot, x')\rangle = K(x, x') \tag{2.24}$$

Hence, we have proved that Definition 2.2 implies the properties defined in Definition 2.1.

In all, the two kernel definitions are equivalent.

Proof ends.

Another theorem, namely Mercer's theorem, correlates these two definitions on the perspective of function analysis. This introduction does not go into details of Mercer's theorem. The elaboration of how Mercer's theorem is used to define kernels can be found in Schölkopf and Smola (2002) and Herbrich (2002).

Now that we have formally presented the definitions of kernel, we will present some more properties of kernel, which may benefit the construction of novel kernels in later chapters. To propose novel positive semidefinite kernel functions is not straightforward. Alternatively, some properties of the kernel functions can be employed to design new kernels.

Proposition 2.3 *Given that* $K_1, K_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *and* $K_3 : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ *are valid kernels, for all* $x, x' \in \mathcal{X}$ *and* $y, y' \in \mathcal{Y}$*, the following satisfies:*

$$1. K(x, x') = K_1(x, x') + K_2(x, x') \quad \textit{is a valid kernel.} \tag{2.25}$$

$$2. K(x, x') = K_1(x, x') K_2(x, x') \quad \textit{is a valid kernel.} \tag{2.26}$$

$$3. K_1 \bigoplus K_2 \left((x, y), (x', y')\right) = K_1(x, x') + K_2(y, y') \quad \textit{is a valid kernel defined on } (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}. \tag{2.27}$$

$$4. K_1 \bigotimes K_2 \left((x, y), (x', y')\right) = K_1(x, x') K_2(y, y') \quad \textit{is a valid kernel defined on } (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}. \tag{2.28}$$

(Proof.) See Shawe-Taylor and Cristianini (2004) for details.

The properties introduced by Proposition 2.3 are able to facilitate the construction of novel kernels based on the predefined valid kernels. These operations may preserve the original characteristics of the predefined kernels and endow the new kernels with more sense.

2.2 Support Vector Machines

Support Vector Machines (SVM) are well known kernel based learning machines (Vapnik, 1998). The key idea of SVM is to find a hyperplane as the decision boundary so that the margin of separation between the positive instances and the negative ones is maximized. The data points with the minimal distance to this hyperplane are called support vectors, with each data point to be represented as a vector from

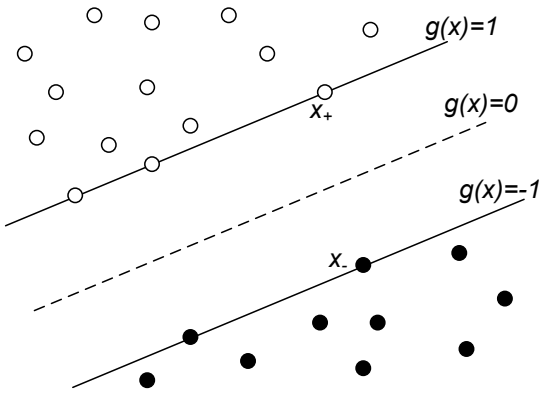


Figure 2.1: Linearly separable
Empty dots refer to positive instances.
Solid dots refer to negative instances.

the origin to this point. Hence, to find the hyperplane can be considered as the problem of finding the support vectors for both positive and negative instances. Vapnik (1998) recognized SVM as an instantiation of the Structural Risk Minimization theory. This theory attributes the classification error on testing data to be bounded by two factors. One is the sum of the training error and the other is complexity of the model, namely Vapnik-Chervonenkis dimensions. In this section, we will give a brief introduction of SVM and see how SVM is related to kernels. We start this section with the simple case that the positive and negative instances are linearly separable.

Linearly separable

Given n training samples $\{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_n, y_n)\}$, where $\vec{x}_i \in \mathbb{R}^N$ is the feature vector of the i-th input data point and $y_i \in \{+1, -1\}$ is the corresponding prediction. It is assumed that the subset of $\{\vec{x}_i, y_i = +1\}$ and the subset of $\{\vec{x}_i, y_i = -1\}$ are linearly separable. As shown in Fig. 2.1, it is possible to construct a separating hyperplane so that all the positive instances are on the one side while all the negative ones are on the other side. In consequence, there exist

$\vec{w} \in \mathbb{R}^N$ and $c \in \mathbb{R}$ satisfying

$$\begin{aligned} \vec{w}^T \vec{x}_i + c \geq 0 \quad \text{for} \quad y_i = +1 \\ \vec{w}^T \vec{x}_i + c < 0 \quad \text{for} \quad y_i = -1 \end{aligned} \tag{2.29}$$

Then, the hyperplane to separate the positive and negative instances is

$$\vec{w}^T \vec{x}_i + c = 0 \tag{2.30}$$

Note that $(\vec{w}, c)$ can be rescaled. For the convenience of solving SVMs, we can define the functional margin of the data set to 1. Therefore, by defining $g(\vec{x}) = \vec{w}^T \vec{x}_i + c$, the data point $(\vec{x}_i, y_i)$ satisfy

$$y_i g(\vec{x}_i) = y_i(\vec{w}^T \vec{x}_i + c) \geq 1 \tag{2.31}$$

As shown in Fig. 2.1, the positive and negative data points are separated by the margin between $g(\vec{x}) = 1$ and $g(\vec{x}) = -1$. This margin can be evaluated by summing up the distance between the hyperplane of $g(\vec{x}) = 0$ and $g(\vec{x}) = 1$ as well as the distance between $g(\vec{x}) = 0$ and $g(\vec{x}) = -1$:

$$\rho = \frac{|g(\vec{x}_+)|}{\|\vec{w}\|} + \frac{|g(\vec{x}_-)|}{\|\vec{w}\|} \tag{2.32}$$

where $g(\vec{x}_+) = 1$ and $g(\vec{x}_-) = -1$. Hence,

$$\rho = \frac{1}{\|\vec{w}\|} + \frac{1}{\|\vec{w}\|} = \frac{2}{\|\vec{w}\|} \tag{2.33}$$

To maximize the margin is equivalent to minimize the Euclidean norm $\|\vec{w}\|$. As a result, the problem of finding the optimal hyperplane is transformed to an optimization problem with convex quadratic objective function and linear constraints.

$$\begin{aligned} \min_{\vec{w},c} \quad & \frac{1}{2}\vec{w}^T \vec{w} \\ s.t. \quad & y_i(\vec{w}^T \vec{x}_i + c) \geq 1 \qquad i = 1, 2, \ldots, n \end{aligned} \tag{2.34}$$

This problem can be solved by introducing n *Lagrange multipliers* $\alpha_i \geq 0$, where $1 \leq i \leq n$. The corresponding Lagrange function is

$$L(\vec{w}, c, \vec{\alpha}) = \frac{1}{2}\vec{w}^T\vec{w} - \sum_{i=1}^{n} \alpha_i(y_i(\vec{w}^T\vec{x}_i + c) - 1) \tag{2.35}$$

where $\vec{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)^T$. By differentiating $L(\vec{w}, c, \vec{\alpha})$ with respect to $\vec{w}$ and c and assigning the results to zero, it yields

$$\vec{w} = \textstyle\sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \tag{2.36}$$

$$0 = \textstyle\sum_{i=1}^{n} \alpha_i y_i \tag{2.37}$$

By applying the above Equations back to the Lagrange function Eq. 2.35, the dual problem of the original primal problem Eq. 2.34 can be formulated as

$$\begin{aligned} \max \quad & \tilde{L}(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \vec{x}_i^T\vec{x}_j \\ s.t. \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \end{aligned} \tag{2.38}$$

The dual problem 2.38 is a standard quadratic optimization problem subjected to linear constraints and can be solved with certain standard quadratic programming libraries, which we do not present in detail. Having determined the optimal $\vec{\alpha}$, the $(\vec{w}, c)$ for the corresponding optimal hyperplane are

$$\vec{w} = \textstyle\sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \tag{2.39}$$

$$c = 1 - \vec{w}^T\vec{x}_+ \tag{2.40}$$

The prediction for an unseen data point $\vec{x}$ is determined by

$$f(\vec{x}) = \text{sign}(\sum_{i=1}^{n} \alpha_i y_i \vec{x}_i^T\vec{x} + c) \tag{2.41}$$

Note that for both formula 2.38 and function 2.41, dot product is evaluated. In the training process, dot product is evaluated between each pair of training

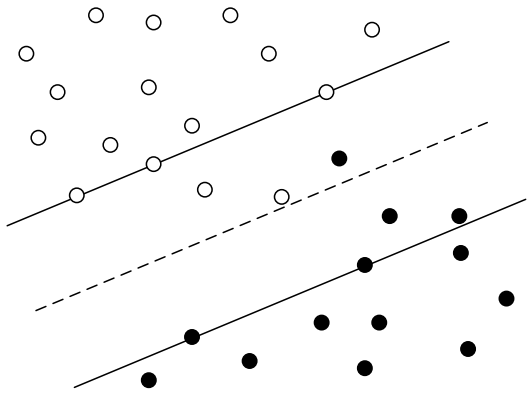


Figure 2.2: Linearly nonseparable

instances in formula 2.38. While testing, dot product is evaluated between the unseen data point and the training instances in function 2.41. From this perspective, kernel methods can be integrated with SVM by replacing the dot product between data points with the kernel function

$$\max \quad \tilde{L}(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\vec{x}_i, \vec{x}_j) \tag{2.42}$$

$$f(\vec{x}) = \text{sign}(\sum_{i=1}^{n} \alpha_i y_i K(\vec{x}_i, \vec{x}) + c) \tag{2.43}$$

where formula 2.38 is rewritten as formula 2.42 and function 2.41 is rewritten as function 2.43.

Linearly nonseparable

By far, we are able to find the optimal hyperplane for the linearly separable data. However, in real application, the training data is quite likely to be linearly nonseparable and it is hard to find a hyperplane to perfectly separate the data in such case. For some cases, even if there is a hyperplane to separate the data, the maximized geometric margin is so small that the model may not generalize well to the unseen data points. In these cases, it is necessary to enlarge the margin to certain extent to make the model tolerant of some classification errors. As shown in Fig. 2.2, some data points are allowed to violate the constraint. This can be

achieved by introducing the slack variables ξ_i for $1 \leq i \leq n$

$$\begin{aligned} \min_{\vec{w},c,\xi_i} \quad & \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=1}^{n}\xi_i \\ s.t. \quad & y_i(\vec{w}^T\vec{x}_i + c) \geq 1 - \xi_i \qquad i = 1, 2, \ldots, n \\ & \xi_i \geq 0 \qquad i = 1, 2, \ldots, n \end{aligned} \tag{2.44}$$

where ξ_i is used to penalize the nonseparable data points. C is the regularization parameter, which is used to balance the training error and the complexity of the model. As C grows larger, the training error rate will decrease. This may lead to a worse generalization to the unseen data. On the contrary, as C grows smaller, the nonseparable data points are less penalized. It may render the model better generalized to the unseen testing data, but less fit to the training data. Therefore, in real application, C should be appropriately selected.

The dual problem of the nonlinear separable case can be formalized as follows:

$$\begin{aligned} \max \quad & \tilde{L}(\vec{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \vec{x}_i^T\vec{x}_j \\ s.t. \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{aligned} \tag{2.45}$$

By using the similar approaches to the linearly separable case, the dual problem in Eq. 2.45 can be solved.

2.3 Preliminary Concepts on Tree Structures

In this section, we will present some basic definitions on parse tree structures. These structures will be adopted throughout the later chapters.

Rooted Labeled Ordered Trees A rooted labeled ordered tree T is a directed acyclic graph, with V as the set of nodes, $E = \{(x, y)|x, y \in V\}$ as the

set of edges. There is a distinguished node $r \in V$ designated as *root* that has no entering edges. Every other node has exactly one entering edge. For all $x \in V$, there is a unique path from r to x. There is a mapping function $\Phi : V \to \Sigma$ to assign every node a label, where $\Sigma = \{\sigma_1, \sigma_2, \dots\}$ is a set of labels. The sibling nodes of the same parent node are ordered, i.e., if a node has p children, we can identify them as $1st$ child, $2nd$ child, ..., p-th child.

The syntactic parse tree is an instantiation of rooted labeled ordered trees. Hereafter, we are dealing with rooted labeled ordered tree, which we may refer as "tree" for short.

Additionally, for a subtree t with its root node n embedded in T, let l be the number of leaf nodes of T. By indexing the leaf node from 0, the span covered by the root node of T is $[0, l-1]$. Furthermore, let $\delta = [t_b, t_e]$ be the span covered by t, with t_b denoting the span index of the leftmost leaf node of t and t_e denoting the span index of the rightmost leaf node of t. In addition, we relate the root node n to δ by denoting $n_b = t_b$ and $n_e = t_e$. Thus, the span covered by n can be written as $\delta = [n_b, n_e]$ as well.

In Fig. 2.3, we give an example of the rooted labeled ordered tree with some structures highlighted. To simplify the illustration, we avoid unimportant nodes and edges in the given tree. The dotted line denotes the path from the node at one end of the line to the node at the other end of the line, where the corresponding nodes and edges on the line are not shown in the figure.

Overlap/Non-Overlap conditions A pair of nodes n and n' is called *non-overlap* if and only if

a. $n_b \neq n'_b$;

b. $n_e \neq n'_e$;

c. if $n_b < n'_b$ then $n_e < n'_b$, while n and n' are interchangeable in this condition.

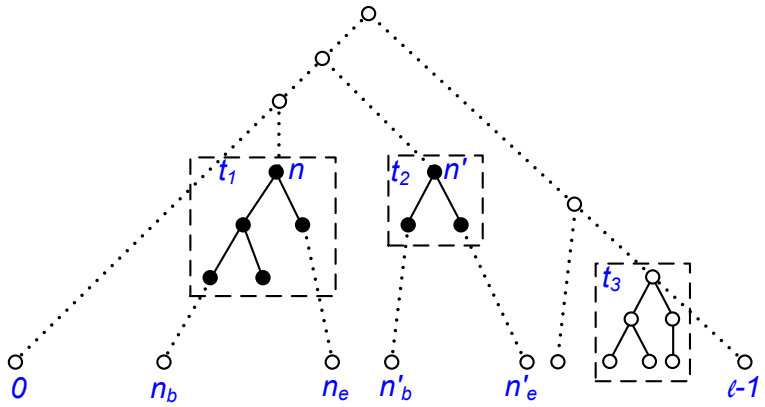


Figure 2.3: Tree Structure Illustration

In addition, a pair of subtrees is called *non-overlap* if and only if the root nodes of the subtrees are *non-overlapped*. For both node pair and subtree pair, if any one of the above conditions does not hold, we call the pair to be *overlapped*. As shown in Fig. 2.3, the nodes n and n' are *non-overlapped*. Correspondingly, the two subtrees t_1, t_2 highlighted with dashed squares are *non-overlapped* as well.

Anchored Subtree Structure An anchored subtree structure is a subtree embedded in a rooted labeled ordered tree. An anchored subtree structure can be manually defined by heuristics according to corresponding tasks. Given a rooted labeled ordered tree, multiple anchored subtrees can be defined in this tree. However, we restrict the anchored subtrees embedded in a rooted labeled ordered tree to be mutually *non-overlapped*. As shown in Fig. 2.3, the subtrees t_1, t_2 can be identified as two anchored subtrees in T.

Subtree Sequence Structure A subtree sequence structure is a sequence of *non-overlapped* subtrees in a rooted labeled ordered tree. A subtree sequence is called *contiguous*, i.e. for all adjacent subtrees t, t' in the sequence, $t_b = t'_e + 1$ or $t'_b = t_e + 1$, or is called *noncontiguous*, i.e. otherwise.

As shown in Fig. 2.3, we can identify $\{t_1, t_2\}$ as a *contiguous* subtree sequence; $\{t_1, t_3\}$ as a *noncontiguous* subtree sequence; $\{t_2, t_3\}$ as a *noncontiguous* subtree

sequence. The above three subtree sequences all consist of two subtrees.[1] In addition, $\{t_1, t_2, t_3\}$ can be identified as a *noncontiguous* subtree sequence. Furthermore, $\{t_1\}$, $\{t_2\}$ and $\{t_3\}$ can be identified as *contiguous* subtree sequences respectively, although each of the sequences only consists of one subtree. Hereafter, we may refer "subtree sequence" to "tree sequence" for simplification.

2.4 Summary

In this chapter, some preliminary knowledge of this thesis is introduced. Specially, positive semidefinite kernels are defined by two prevalent definitions. We present that the two widely used definitions are actually equivalent. Hence, when a kernel is constructed by one definition, the conditions in the other definitions can be used as properties for the kernel. In addition, Support Vector Machines (SVM) are introduced as well. SVM is a kernel machine, in which the proposed kernels in Chapter 4 will be integrated. We also include some definitions for special structures on the rooted labeled ordered trees, since these structures will recur throughout the whole thesis.

[1] When we mention the subtree in a subtree sequence, we refer to the subtree structure which is not a part of other subtrees in the given tree sequence, i.e. not connected to any other nodes in the tree sequence. This note is used to distinguish the fact that small subtrees can be extracted from large subtrees by ignoring certain edges and nodes.

Chapter 3

Kernels on Discrete Structures

In the last chapter, we have introduced some basis of kernel methods. Specially, we have explained that kernels are able to conveniently employ the input data with complex structures without explicit feature engineering to construct the vectorial representation. The complex input structures in NLP can be sequences, trees and graphs. In this chapter, we introduce some representative works which apply kernel methods in these complex structures. On this perspective, feature vectors of the input data are composed of the weights of these discrete elementary structures. By applying kernel methods in these complex input structures, rich structure features can be well captured. These features have been verified to be effective in various applications in the literature of NLP, Bioinformatics and Web Mining.

This chapter mainly serves as a literature review. In Section 3.1, some representative work are elaborated. Specifically, we review convolution kernels proposed by Haussler (1999) in Section 3.1.1. Convolution kernel can be considered as the fundamental theory for a variety of kernels on discrete structures. In Section 3.1.2, we discuss a more general form of convolution kernel, namely mapping kernel (Shin and Kuboyama, 2008). After that, we review two representative kernels on discrete structures, i.e. sequence kernel (Section 3.1.3) and Collins and Duffy's tree kernel

(Section 3.1.4) respectively. In the second part of this chapter, more related works that apply kernels in NLP applications are reviewed.

3.1 Representative Kernels

Basically, in this section we will review Haussler's convolution kernel as well as its applications. Haussler (1999) presented a convenient paradigm to construct novel kernels based on the predetermined kernels on the substructures. This can save the effort of constructing kernels from scratch, which requires to prove the positive semidefinite property of the new kernels. Later, Shin and Kuboyama (2008) proposed another paradigm to construct new kernels, namely mapping kernel, which can be considered as the generalization of convolution kernels. Mapping kernel can be used to construct additional novel kernels by applying certain constraints on the substructure space inherited from the corresponding convolution kernel through a mapping system. Shin and Kuboyama (2008) also provided a necessary and sufficient condition to equip mapping kernel with the positive semidefinite property.

In real applications, convolution kernel can be applied in various structures, such as sequences, trees and graphs. If a symmetric function evaluated on such discrete structures is constructed by the paradigms of convolution kernel or mapping kernel, the function is guaranteed to be a valid positive semidefinite kernel. Alternatively, the validity of a positive semidefinite kernel can be proved by using Definition 2.1 as a necessary and sufficient condition. By using this property, it is necessary to construct a mapping function to project the input data to a feature vector that is composed of the concerned substructures in the dot product space. If such a map is found, the function is a valid kernel as well.

In addition to the paradigms to construct valid kernels, we also elaborate two representative kernels that will facilitate the understanding of Chapter 4.

3.1.1 Haussler's Convolution Kernel

As previously stated, to verify the positive semidefinite property for a kernel function is not straightforward in certain cases. Therefore, it may be difficult to build up a kernel from scratch due to this requirement. Haussler (1999) proposed a general framework to construct new positive semidefinite kernels on discrete structures. The key idea of this approach is to employ the already verified positive semidefinite kernels defined on the feature space of which elements share certain relationship with the original structures of the input data. First, it is required to decompose an original input structure into substructures based on the given relationship. Second, the predefined valid kernels are evaluated on the corresponding substructure space respectively. Finally, kernels defined on the original input structure can be evaluated in a convolution manner by summing up the underlying predefined kernels over all the different decompositions.

Definition 3.1 [**Convolution Kernel**] *Let $x, x' \in \mathcal{X}$ be the structure of the input data. Let $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_D$ be the nonempty separable metric spaces. For an input structure x, let $\vec{x} = (x_1, x_2, \ldots, x_D)$ be "parts" of x, where $x_d \in \mathcal{X}_d$ for $1 \leq d \leq D$. Let $k_d : \mathcal{X}_d \times \mathcal{X}_d \to \mathbb{R}$ be a valid underlying kernel for the d-th metric space. Given a relation $R \subseteq (\mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_D) \times \mathcal{X}$ and the set $\mathcal{S} = \{x : \exists\, \vec{x}, (\vec{x}, x) \in R\}$, the function $K' : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ defined in Eq. 3.1 is a valid kernel.*

$$K'(x, x') = \sum_{(\vec{x},x)\in R} \sum_{(\vec{x}',x')\in R} \prod_{d=1}^{D} k_d(x_d, x'_d) \tag{3.1}$$

The validity of this kernel is proved in Theorem 1 in Haussler (1999). K' can be further extended from the set $\mathcal{S}$ to the entire input structure space $\mathcal{X}$. Since $\mathcal{S} \subseteq \mathcal{X}$, the extension of kernel K' can be achieved by defining the kernel to be 0 if $x \notin \mathcal{S}$ or $x' \notin \mathcal{S}$. It is easy to verify that the extended kernel is also a valid positive semidefinite kernel. As a result, we can obtain the convolution kernel

$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ based on this extension:

$$K(x, x') = \sum_{\substack{x \in \mathcal{S} \\ (\vec{x}, x) \in R}} \sum_{\substack{x' \in \mathcal{S} \\ (\vec{x}', x') \in R}} \prod_{d=1}^{D} k_d(x_d, {x_d}') \tag{3.2}$$

To construct a kernel by the convolution paradigm, it requires to appropriately define a relation R by setting up a scheme that how an input structure can be decomposed. In addition, we also need to employ the underlying valid kernels $k_d : \mathcal{X}_d \times \mathcal{X}_d \to \mathbb{R}$ for each of the metric space. Based on the definition of relation R and the underlying kernels k_d for $1 \leq d \leq D$, we may design various kernels without being bothered by the proof of the positive semidefinite property. Some examples of relation R are provided in Section 2.2 of Haussler (1999). Additionally, we will give an illustration on how sequence kernel (Section 3.1.3) and Collins and Duffy's tree kernel (Section 3.1.4) can be constructed based on this convolution paradigm later in this chapter.

3.1.2 Mapping Kernel

From Eq. 3.2, we can observe that the convolution kernel paradigm constructs novel kernels by traversing the entire cross product of the possible decompositions of the original structures. It is possible that some decompositions yield ineffective substructures for the concerned task. Therefore, discarding these substructures may facilitate other effective substructures to play a more important role in the model. Consequently, Shin and Kuboyama (2008) proposed mapping kernel by restricting the explored space to be a subset of the cross product of the entire substructure space.

Definition 3.2 [**Mapping Kernel**] *Let* $x, x' \in \mathcal{X}$ *be the input data. Let* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *be a valid kernel. Then Eq. 3.3 is a valid positive semidefinite kernel*

$$K(x, x') = \sum_{(u,u') \in \mathcal{M}_{x,x'}} k(u, u') \tag{3.3}$$

when the following two conditions satisfy

(1) $\mathcal{M}$ *is a finite and symmetric set, which is defined by a mapping system* $\mathbb{M}$

$$\mathbb{M} = (\mathcal{X}, \{\mathcal{U}_x | x \in \mathcal{X}\}, \{\mathcal{M}_{x,x'} \subseteq \mathcal{U}_x \times \mathcal{U}_{x'} | (x, x') \in \mathcal{X} \times \mathcal{X}\}) \tag{3.4}$$

(2) $\mathbb{M}$ *is transitive, such that for all* $x_1, x_2, x_3 \in \mathcal{X}$

$$(u_1, u_2) \in \mathcal{M}_{x_1,x_2} \wedge (u_2, u_3) \in \mathcal{M}_{x_2,x_3} \Rightarrow (u_1, u_3) \in \mathcal{M}_{x_1,x_3} \tag{3.5}$$

Specifically, the second component of the triplet $\mathbb{M}$, i.e. $\mathcal{U}_x$, is the substructure space of the input data, while the third component can be considered as the model related feature space which is the subset of cross product of the entire substructure space. Given the mapping system in Eq. 3.4, the property of $\mathbb{M}$ to be transitive is the necessary and sufficient condition for Eq. 3.3 to be a valid kernel.

Mapping kernel is a generalization of convolution kernel, since the former turns out to be the latter if $\mathcal{M}_{x,x'} = \mathcal{U}_x \times \mathcal{U}_{x'}$ satisfies. In addition, mapping kernel provides a more flexible and convenient paradigm to construct novel kernels. We will demonstrate the advantage of mapping kernel by comparing mapping kernel and convolution kernel with some examples.

Given $x, x' \in \mathbb{R}^N$, where x can be written with a vector $\vec{x} = (x_1, x_2, \ldots, x_N)^T$, we define two symmetric functions $k_1, k_2 : (\mathbb{R} \times \mathbb{N}) \times (\mathbb{R} \times \mathbb{N}) \to \{0, 1\}$ and prove k_1 and k_2 are both kernels.

$$k_1\left((x_i, i), (x'_j, j)\right) = \begin{cases} 1 & \text{if } x_i = x'_j \text{ and } i = j \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

$$k_2\left((x_i,i),(x'_j,j)\right) = \begin{cases} 1 & \text{if } x_i = x'_j \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

Proposition 3.1 *k_1 and k_2 are positive semidefinite kernels.*

Proof. We first prove k_2 is a valid kernel.

Given that $k_2 : (\mathbb{R}\times\mathbb{N})\times(\mathbb{R}\times\mathbb{N}) \to \{0,1\}$ is a symmetric function, we define a map $\mathbf{\Phi} : (\mathbb{R}\times\mathbb{N}) \to \{0,1\}^\infty$ such that for all $(x_i,i) \in \mathbb{R}\times\mathbb{N}$

$$\begin{aligned} \mathbf{\Phi}(x_i,i) &= (\Phi_1((x_i,i)),\Phi_2((x_i,i)),\ldots)^T \\ \text{where } \Phi_m((x_i,i)) &= \begin{cases} 1 & \text{if } x_i = m \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.8}$$

Based on this map, k_2 can be rewritten as

$$\begin{aligned} k_2\left((x_i,i),(x'_j,j)\right) &= \langle \mathbf{\Phi}(x_i,i), \mathbf{\Phi}(x'_j,j)\rangle \\ &= \sum_{m=1}^{\infty} \Phi_m(x_i,i)\Phi_m(x'_j,j) \\ &= \begin{cases} 1 & \text{if } x_i = x'_j \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.9}$$

Hence, we have proved that k_2 is a valid kernel by Definition 2.1.

To prove k_1 is a valid kernel, we need another underlying kernel $k_3 : \mathbb{R}\times\mathbb{R} \to \{0,1\}$, such that given $y, y' \in \mathbb{R}$

$$k_3(y,y') = \begin{cases} 1 & \text{if } y = y' \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

which can be easily proved to be a valid kernel using the same method as k_2. Then, based on Property 2.28 in Proposition 2.3, k_1 can be rewritten as the tensor product

of two valid kernels

$$k_1\left((x_i,i),(x'_j,j)\right) = k_3(x_i,x'_j)k_3(i,j)$$
$$= \begin{cases} 1 & \text{if } x_i = x'_j \text{ and } i = j \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

Therefore, we have proved k_2 is a valid kernel as well.

Proof ends.

By means of the underlying kernels k_1 and k_2, we can construct a novel kernel to compute the similarity of the vectorial representation of x and x'. This kernel compares each pair of component x_i, x'_i for all $0 \leq i \leq N$ and returns 1 if $x_i = x'_i$, while 0 otherwise. Then the kernel value is just the sum of all the pairwise component similarity scores.

To construct this kernel based on the convolution kernel paradigm, we first define a relation $R = ((x_i,i),x)$ for all $x \in \mathbb{R}^N$ and $1 \leq i \leq N$. Hence, $K_1 : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ is a kernel such that

$$\begin{aligned} K_1(x,x') &= \sum_{((x_i,i),x)\in R} \sum_{((x'_j,j),x')\in R} k_1((x_i,i),(x'_j,j)) \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} k_1((x_i,i),(x'_j,j)) \end{aligned} \tag{3.12}$$

Alternatively, based on the mapping kernel paradigm, the same kernel can be written as Eq. 3.13 using k_2 as the underlying kernel

$$K_2(x,x') = \sum_{\substack{i=1 \\ i=j}}^{N} k_2((x_i,i),(x'_j,j)) \tag{3.13}$$

From Eq. 3.12, we can observe that K_1 sums up the entire cross product $R \times R$ based on the underlying kernel k_1. By contrast, Eq. 3.13 restricts the set of elements summed up by the constraint $i = j$ based on the underlying kernel k_2. By applying

the relation R defined in Eq. 3.12 together with the constraint $i = j$, K_2 corresponds to the convolution kernel paradigm in such a manner that

$$K_2(x, x') = \sum_{((x_i,i),x)\in R} \sum_{\substack{((x'_j,j),x')\in R \\ i=j}} k_2((x_i, i), (x'_j, j)) \tag{3.14}$$

It is clear to see from Eq. 3.14 that K_2 only considers a subset structures of the entire cross product space covered by convolution kernel. This is also the key advantages of the mapping kernel paradigm over the convolution kernel paradigm. When constructing a novel kernel with such constraints, mapping kernel is able to leave the constraint to be incurred outside the underlying kernel at the summation phase. On the contrary, the convolution kernel paradigm has to integrate the constraint in the underlying kernels. This may lead to more complex underlying kernels, whose positive semidefinite property is difficult to be verified.

In addition, some constraints can only be dealt with outside the underlying kernels so that the proposed kernels cannot be constructed using the convolution paradigm. Consider the following example constructed under the mapping kernel paradigm,

$$K'_2(x, x') = \sum_{\substack{i=1 \\ i\leq j}}^{N} k_2((x_i, i), (x'_j, j)) \tag{3.15}$$

By the transitive condition 3.5 in Definition 3.2, it is easy to prove Eq. 3.15 is a valid kernel, since the operator $\leq$ is transitive. However, when attempting to construct this kernel under the convolution kernel paradigm, it is not straightforward to integrate the constraint into any underlying kernel. In fact, it seems infeasible to find a mapping function that can obtain a projected image for x without considering x' under this constraint. Instead, the mapping kernel paradigm applies the constraint outside the underlying kernel so that the proposed kernels with complex constraints can be easily constructed.

In conclusion, mapping kernel is advantageous over convolution kernel in its flexibility to handle the constraints. Generally, convolution kernel can be considered as a special case of mapping kernel. In case no constraint is applied so that mapping kernel needs to go over the entire cross product of substructures, mapping kernel turns out to be equivalent to convolution kernel.

3.1.3 Sequence Kernel

Sequence kernel applied in document classification explores the feature space with respect to bag of characters (Lodhi et al., 2002) and bag of words (Cancedda et al. 2003). Given two sequences s and s', sequence kernel match all identical subsequences shared by s and s'. The kernel function is then evaluated by the number of matched subsequences (both contiguous and non-contiguous).

Let Σ be a finite symbol set. Let $s = s_0 s_1 \cdots s_{|s|-1}$ be a sequence with each item s_i to be a symbol, i.e. $s_i \in \Sigma, 0 \leq i \leq |s| - 1$. Let $\mathbf{i} = [i_1, i_2, \ldots, i_m]$, where $0 \leq i_1 < i_2 < \cdots < i_m \leq |s| - 1$, be a subset of indices in s. Let $s[\mathbf{i}] \in \Sigma^m$ refer to the subsequence of $s_{i_1} s_{i_2} \cdots s_{i_m}$.

Given the m-length subsequence space $\mathcal{S}_m = \{u_1, \ldots, u_i, \ldots, u_{|\mathcal{S}|}\}$, where $u_i \in \Sigma^m$ for all $1 \leq i \leq |\mathcal{S}|$, it requires to construct a feature vector for the given sequence s, using the integer counts of the corresponding m-length subsequences. Hence, the m-length subsequence vector of s can be expressed as follows:

$$\mathbf{\Phi}_m(s) = \big(\#(u_1), \ldots, \#(u_i), \ldots, \#(u_{|\mathcal{S}|})\big)^T \tag{3.16}$$

where $\#(u_i)$ is the number of occurrences of the subsequence type u_i.

In addition, for a subsequence u and a subset of indices $\mathbf{i}$ for a given sequence s, let $I(u, \mathbf{i})$ refers to the indicator function which equals to 1 if and only if $u = s[\mathbf{i}]$ and 0 otherwise. Then for a certain subsequence length of m, the kernel function

is defined in Eq. 3.17.

$$\widetilde{K}_m(s,s') = \langle \mathbf{\Phi}_m(s), \mathbf{\Phi}_m(s') \rangle \tag{3.17a}$$

$$= \sum_{\substack{u\in\Sigma^m \\ u'\in\Sigma^m \\ u=u'}} \left(\sum_{\mathbf{i}:u=s[\mathbf{i}]} I(u,\mathbf{i}) \cdot \sum_{\mathbf{j}:u'=s'[\mathbf{j}]} I(u',\mathbf{j}) \right) \tag{3.17b}$$

$$= \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u'=s'[\mathbf{j}]} \sum_{\substack{u\in\Sigma^m \\ u'\in\Sigma^m \\ u=u'}} (I(u,\mathbf{i}) \cdot I(u',\mathbf{j})) \tag{3.17c}$$

From Eq. 3.17c, we can detect that $\sum_{\substack{u\in\Sigma^m \\ u'\in\Sigma^m \\ u=u'}} (I(u,\mathbf{i}) \cdot I(u',\mathbf{j}))$ is a valid underlying kernel, since it is actually a dot product of vectors defined in Eq. 3.16. As a result, $\widetilde{K}_m(s,s')$ can be considered as an instantiation of convolution kernel.

The standard sequence kernel (Lodhi et al., 2002; Cancedda et al., 2003) introduces a weighting function to weight the subsequence structures as follows:

$$\widetilde{K}_m(s,s') = \sum_{\substack{u\in\Sigma^m \\ u'\in\Sigma^m \\ u=u'}} \left(\sum_{\mathbf{i}:u=s[\mathbf{i}]} p(u,\mathbf{i}) \cdot \sum_{\mathbf{j}:u'=s'[\mathbf{j}]} p(u',\mathbf{j}) \right) \tag{3.18}$$

where $p(u,\mathbf{i})$ and $p(u',\mathbf{j})$ are the respective weights contributed to the kernel value when u and u' are matched.

Specially, Lodhi et al. (2002) and Cancedda et al. (2003) employ a weighting function which penalizes both the number of *matching symbols* and the *length of gaps* using the *same* decay factor $\rho \in [0,1]$. In other words, the penalty covers the whole span of a given subsequence from the very beginning symbol to the ending symbol.

$$\begin{aligned} p(u,\mathbf{i}) &= \rho^{(i_m - i_1 + 1)} \\ p(u',\mathbf{j}) &= \rho^{(j_m - j_1 + 1)} \end{aligned} \tag{3.19}$$

3.1.4 Collins and Duffy's Tree Kernel

Collins and Duffy's Tree kernel (Collins and Duffy, 2002) is a special instantiation of convolution kernel for discrete structures (Haussler, 1999). Given a parse tree, the syntactic features are defined as all types of subtrees and each feature value is the number of the occurrences of the corresponding subtree. As shown in Fig 3.1, the feature space of the parse tree structure over "last year 's" is a set of 11 subtree types with each type of subtree appeared once.

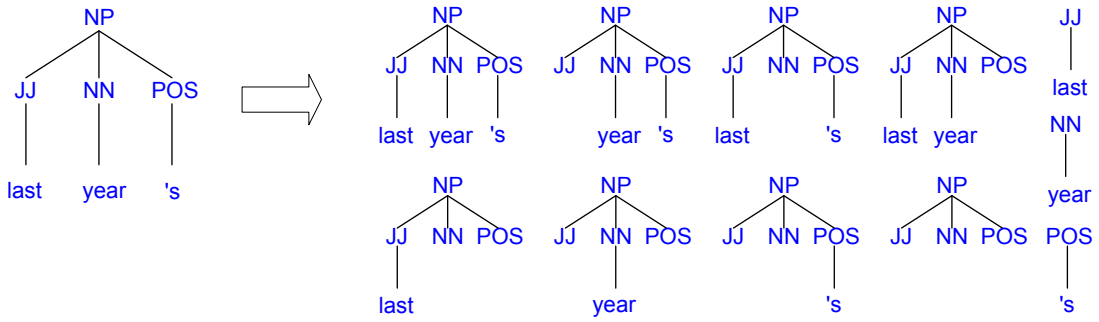


Figure 3.1: Collins and Duffy's Tree Kernel feature space.

Traditional feature based methods explicitly represent the features with a feature vector. Given the substructure space $\mathcal{T} = \{t_1, \ldots, t_i, \ldots, t_{|\mathcal{T}|}\}$, feature based methods construct a feature vector for the given parse tree T, using the integer counts of the corresponding subtrees. Hence, T can be expressed as follows:

$$\mathbf{\Phi}(T) = \big(\#(t_1), \ldots, \#(t_i), \ldots, \#(t_{|\mathcal{T}|})\big)^T \tag{3.20}$$

where $\#(t_i)$ is the number of occurrences of the subtree type t_i.

However, it may be computationally infeasible to explicitly construct such a feature vector, since the number of subtree types is exponential with the tree size. Alternatively, convolution tree kernel computes the number of common subtrees between two parse trees by implicitly evaluating the dot product in the high dimensional feature space without explicitly enumerating the features.

The kernel function is evaluated in Eq. 3.21. T and T' refer to the parse tree pair to be evaluated. N_T and $N_{T'}$ refer to the node sets of T and T' respectively. $I_i(n)$ refers to an indicator function which equals to 1 if and only if the sub-structure t_i is rooted at the node n and 0 otherwise. $\Lambda(n, n') = \sum_{i=1}^{|\mathcal{T}|} (I_i(n) \cdot I_i(n'))$ is the total number of identical subtrees rooted at n and n'.

$$K_t(T, T') = \langle \mathbf{\Phi}(T), \mathbf{\Phi}(T') \rangle \tag{3.21a}$$

$$= \sum_{i=1}^{|\mathcal{T}|} \left(\sum_{n \in N_T} I_i(n) \cdot \sum_{n' \in N_{T'}} I_i(n') \right) \tag{3.21b}$$

$$= \sum_{n \in N_T} \sum_{n' \in N'_T} \sum_{i=1}^{|\mathcal{T}|} (I_i(n) \cdot I_i(n')) \tag{3.21c}$$

$$= \sum_{n \in N_T} \sum_{n' \in N'_T} \Lambda(n, n') \tag{3.21d}$$

From Eq. 3.21d, we can observe that $K_t(T, T')$ is constructed by the convolution kernel paradigm since $\Lambda(n, n')$ is a valid underlying kernel. For $\Lambda(n, n')$ to be a valid kernel, it can be directly shown to be the dot product of vectors defined in Eq. 3.20 as presented in 3.21c.

Then convolution tree kernel follows the dynamic programming scheme to calculate $\Lambda(n, n')$ as follows:

Algorithm. Evaluation of $\Lambda(n, n')$ function

- **if** the production rule at n and n' are different,

$$\Lambda(n, n') = 0, \tag{3.22a}$$

- **else if** both n and n' are pre-terminal nodes

$$\Lambda(n, n') = \lambda, \tag{3.22b}$$

- **else**

$$\Lambda(n, n') = \lambda \prod_{j=1}^{nc(n)} (1 + \Lambda(c(n, j), c(n', j))), \tag{3.22c}$$

where $nc(n)$ is the number of child nodes of n; $c(n, j)$ is the j-th child of n; λ is the decay factor for the depth of the subtree, which penalizes the large structures with respect to the size of the subtree. By implementing the kernel using the above algorithm, the evaluation of convolution tree kernel requires a simultaneous post-order traverse of T and T', which can be simulated by incrementally filling in an $|N_T| \cdot |N_{T'}|$ matrix. Therefore, the computational complexity of convolution tree kernel is $O(|N_T| \cdot |N_{T'}|)$.

3.2 Previous Work

Kernels over tree structures have been widely exploited and been verified to improve the performance in various NLP applications.

For phrase based trees, efforts are endeavored to deeply explore the syntactic features and extend the feature space beyond the subtree representation exploited by Collins and Duffy's tree kernel. One of the pioneer work in the parse tree domain is Partial Tree Kernel (PTK) (Moschitti, 2006). PTK allows partial production rule matching within a syntactic structure, which can extensively enlarge the feature space. An example is given in Fig. 3.2 to illustrate the partially matched tree structures. Nevertheless, the idea of partial rule matching is to ignore some child nodes and not rely on the original production rules as a hard constraint. Experimental results suggest that PTK may compromise the performance compared with Collins and Duffy's tree kernel for the task of question classification in phrase parse tree. In addition, their experimental results also reveal that PTK can be more effective on dependency structures.

To overcome the non-grammatical issue in PTK, Zhang et al. (2008a) proposed

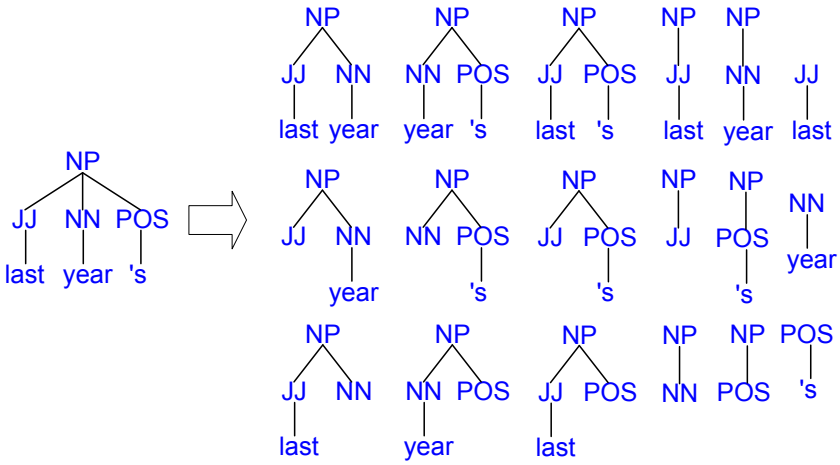


Figure 3.2: Illustration of feature space of Partial Tree Kernel (PTK)

grammar driven tree kernel to explore a larger substructure space by modifying the original subtree structure and generalizing certain grammar tags. Unlike PTK, which still requires exact rule matching, this kernel allows fuzzy matching between grammatically similar rules. However, the modification of the tree structure is based on certain grammar rules. This may require human effort to analyze and select the beneficial set of grammatical representations and may make the method difficult to extend to other languages and grammars. Zhang, Zhang, and Li (2010) proposed Forest Kernel to address the parsing error problem, which explores a larger feature space from the perspective of capturing more accurate tree structures. Since most tree kernels explore the substructure on the single 1-best parse tree generated by the automatic parser, the parsing errors and data sparseness may largely compromise the performance of the kernels. Therefore, introducing more parse tree candidates may alleviate those problems. By exploring the substructures on the packed forest, which is a compact representation to encode exponentially large number of trees using a hypergraph, substructures embedded in the n-best parse trees can be efficiently mined and weighted. The effectiveness of these additional substructures from the non-1-best parse tree is verified by relation extraction and semantic role

labeling.

For kernels on dependency structures, Zelenko, Aone, and Richardella (2003) proposed a recursively matching kernel on shallow parse trees for relation extraction. The kernel recursively conducts the structure matching layer by layer in a top down manner. For each node, it matches the subsequences of its child nodes, either contiguous or non-contiguous. Culotta and Sorensen (2004) further applied this kernel on the augmented dependency parse trees. More semantic and syntactic information is integrated into each node of the dependency tree other than the words. As a result, this kernel not only matches the subsequence of the child nodes for a given node pair, it also matches the feature vectors embedded in the node pair and computes a soft matching score for them. Unlike Collins and Duffy's tree kernel and PTK, which equally consider the substructures in a parse tree, the previous two kernels would rather require the hierarchical structures of the parse trees to be consistent, since the matchable nodes are required to be in the same layer. Bunescu and Mooney (2005) proposed another dependency tree kernel to count the number of common word classes at each position in the shortest paths between two entities. It is required that the length of the matched paths should be the same. Otherwise the two paths are considered as unmatchable. These two dependency kernels are further modified by Reichartz, Korte, and Paass (2009), who extended the first dependency kernel (Culotta and Sorensen, 2004) by allowing the matching of every combination of nodes in the given tree pair and extended the second one (Bunescu and Mooney, 2005) by allowing all possible subsequences of nodes along the shortest paths to be matched. The relaxation of the hard constraints of the previous kernels benefits the task of relation extraction, especially improves the recall. Similar with Bunescu and Mooney (2005), Kate (2008) proposed kernels to evaluate the common dependency paths. However, unlike the former work to evaluate the common paths between a pair of predefined nodes (entities), the latter

work finds all the common paths between two parse trees. This extension makes the common paths based dependency kernels more attractive for tasks other then relation extraction.

For other tree structures, Shen, Sarkar, and Joshi (2003) applied kernel methods on LTAG based tree structure for parse tree reranking. The kernel requires to obtain a LTAG derivation tree for each parse tree before evaluating the kernel. Kashima and Koyanagi (2002) presented kernels on labeled ordered trees and embedding trees. The labeled ordered tree kernel allows the mutation of the child nodes given a root node. In addition, embedding trees across multiple layers are also matched. The proposed kernel well facilitates HTML document analysis. However, similar with PTK, the labeled order tree kernel is not grammar driven. Thus, when applied in NLP applications, it may over-generate the non-grammatical substructures.

The above discussions suggest that effective tree kernels are often proposed by extending the feature space of previous kernels with mineaningful substructures. For example, PTK (Moschitti, 2006) is developed based on Collins and Duffy's tree kernel by allowing non-grammatical rule matching. Grammar driven tree kernel (Zhang et al., 2008a) extending Collins and Duffy's tree kernel by allowing fuzzing matching of grammatical rules. The dependency paths kernel is extended from the paths between entity pairs (Bunescu and Mooney, 2005) to all the paths on the parse trees (Kate, 2008) and from contiguous paths (Bunescu and Mooney, 2005) to the non-contiguous node sequences along the path (Reichartz, Korte, and Paass, 2009). Based on the above observation, this thesis attempts to extend the general subtree feature space of Collins and Duffy's tree kernel in a innovative perspective by grouping multiple non-overlapping subtrees as a single feature representation. The additional substructure features, namely subtree sequences, are expected to overcome certain limitations of the single tree representation and bring

meaningful substructures to benefit NLP tasks. The details of the motivation, definition and implementation of the subtree sequence based kernels will be elaborated in Chapter 4.

3.3 Summary

In this chapter, we have reviewed some related work, from which we observe that new kernels can be designed by introducing more meaningful substructures to the current feature space. In addition, two paradigms to construct kernels on discrete structures have been discussed. Specially, mapping kernel is the generalization of convolution kernel and is more flexible with constraints on the substructure space. Two instantiations of convolution kernel, i.e. sequence kernel and Collins and Duffy's tree kernel are elaborated, which will be referred to in the later chapters.

Chapter 4

Tree Sequence based Kernels

The previous chapters have provided the reader with the basic notions of kernel methods and the approaches of constructing kernels on discrete structures such as the convolution kernel paradigm (Section 3.1.1) and the mapping kernel paradigm (Section 3.1.2). For the discrete input structures in NLP, such as trees and sequences, kernels can be constructed by both approaches on the corresponding structures, i.e. tree kernels (Section 3.1.4) and sequence kernels (Section 3.1.3). The advantages of these structure features have been demonstrated on many NLP applications.

In view of the success of tree kernels and sequence kernels, we attempt to combine the advantages of both kernels by exploring the substructure space of tree sequences. The tree sequence structure may provide more meaningful structure features and alleviate certain weakness of the single tree based features. Furthermore, we attempt to apply the tree sequence based kernels on multiple parse trees. It is expected that these extensions can further benefit certain NLP applications.

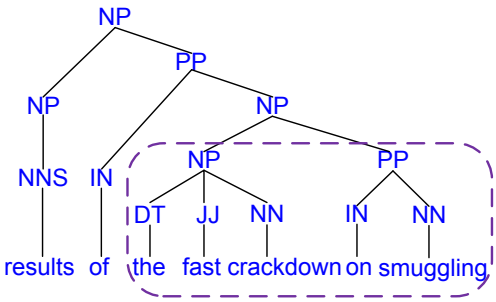
Accordingly, this chapter is organized as follows. In Section 4.1, we introduce the motivation of the tree sequence kernels. Then, we propose contiguous Tree Sequence Kernel (cTSK) in Section 4.2, which is considered as a special case of the

generalized Tree Sequence Kernel (TSK). Before proposing Tree Sequence Kernels (TSKs) (Section 4.4), we propose Set Sequence Kernels (SSKs) in Section 4.3 to faciliate the evaluation of TSKs. In Section 4.5, the extensions of TSKs are proposed, namely Anchored Tree Sequence Kernels (aTSKs), which may facilitate the tasks of modeling the target constituents in the parse trees. In the end, we propose approaches to apply the tree (sequence) based kernels on multiple parse trees.

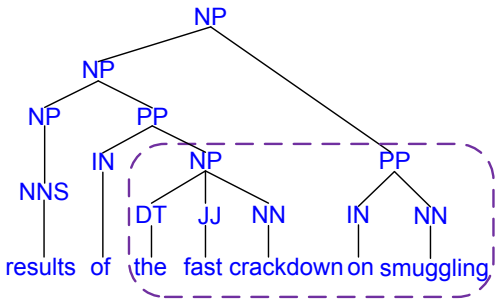
4.1 From Tree Kernel to Tree Sequence Kernel – Some Motivating Examples

Witnessing the effectiveness of tree kernels in many applications, we propose a series of kernels which employ a sequence of subtrees as features. In the tree sequence based kernels, the number of common subtree sequences is counted as the feature value for each subtree sequence type. Before proposing the kernel functions, we illustrate the motivations of the proposed kernels in this section to facilitate an intuitive understanding.

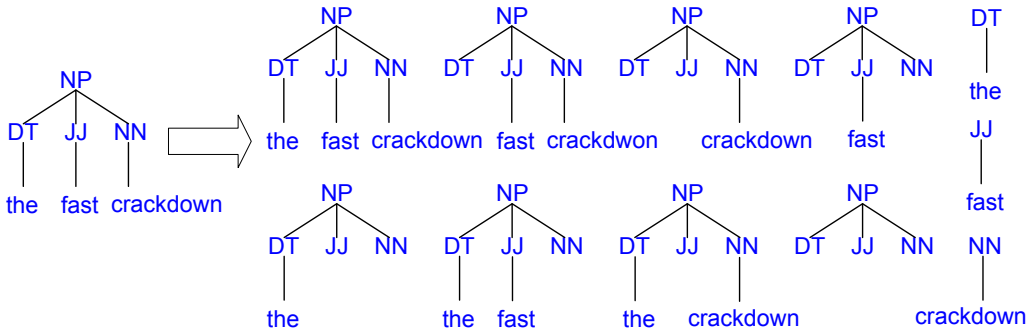
Tree kernel requires the matched substructure to be exactly one single subtree. This constraint may be too strict to sufficiently explore the structure features. First, the single subtree constraint cannot capture certain useful patterns that are not covered by a single subtree. For example, in the parse tree shown in Fig. 4.1(a), it is very likely that the pattern of "*results of the fast crackdown*" is an informative pattern. However, the context is not covered by a single subtree. Therefore, this pattern cannot be captured by the single tree based kernels. Second, to match a pattern covering a rich context with lexical leaf nodes, tree kernel generally requires a large structure equipped with many nodes and much height. However, matching such a large structure may suffer from the data sparseness issue so that tree kernel may not effectively capture large structures. Third, parse trees



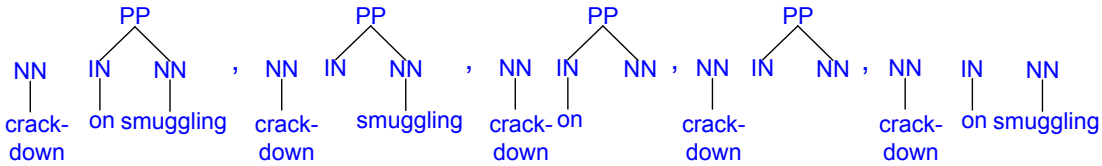
(a) Parse Tree Exp. 1



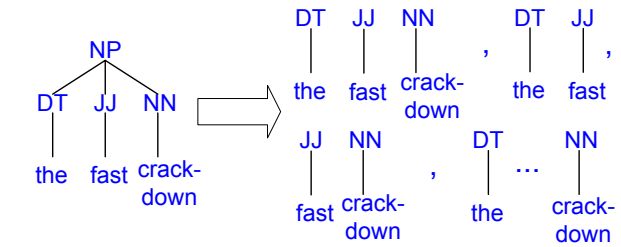
(b) Parse Tree Exp. 2



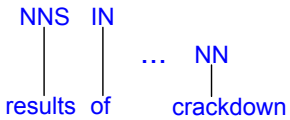
(c) Single subtree features for a simple structure



(d) Contiguous tree sequence features on span $[4, 6]$ of Exp.1



(e) Tree sequence features for a simple structure, excluding single subtree features



(f) Non-contiguous subtree sequence features on span $([0, 1],[4, 4])$

Figure 4.1: Illustration of Tree Sequence Structures.

are usually generated by automatic syntactic parsers. Therefore, parsing errors are inevitable so that the performance of the kernels evaluating the error prone parse trees will compromise. For example, two parse trees for the context of "*results of the fast crackdown on smuggling*" are presented in Fig. 4.1(a) (correct) and Fig. 4.1(b) (with parsing errors). It is easy to see that the syntactic structure over the context of "*the fast crackdown on smuggling*" in Fig. 4.1(b) is not correctly parsed as a single subtree. Therefore, it is infeasible to capture the syntactic features covering this span as a whole from the incorrect parse tree.

The structure of a subtree sequence embedded in a parse tree is an arbitrary number of *non-overlapping* subtrees. We present in Fig. 4.1(c-f) some examples of the subtree sequence structures obtained from the parse tree in Fig. 4.1(a). On the one hand, when the number of subtrees in the subtree sequence is limited to 1, the subtree sequence structure is equivalent to the single subtree structure. In other words, the single subtree structure is a special case of the subtree sequence structure. As shown in Fig. 4.1(c), eleven single subtree features can be captured by Collins and Duffy's tree kernel for the tree over "*the fast crackdown*". On the other hand, when the number of subtrees in the subtree sequence is unlimited, the additional features of multiple subtrees as a single structure can be captured by subtree sequence based kernels in Fig. 4.1(e). The subtree sequence can either be contiguous, i.e. covering a contiguous context as shown in Fig. 4.1(d) or non-contiguous, i.e. with gaps between the adjacent subtrees (Fig. 4.1(f)).

In Fig. 4.1, it can be observed that the subtree sequence based kernels may be advantageous over the single subtree based kernels from the following aspects.

First, the subtree sequence based kernels can capture additional useful patterns other than the single subtree structure. As discussed above, the context of "*results of*" in Fig. 4.1(a) is not covered by a single tree. Alternatively, the span is covered by a subtree sequence with two subtrees so that the syntactic structures over "*results*

of" can be captured by the subtree sequence based kernels.

Second, the subtree sequence based kernels alleviate the data sparseness issue of matching large structures by decomposing a large structure into multiple disconnected parts and matching certain parts as a single structure. For example, to capture the pattern covering the context "*results of*" and "*crackdown*" in Fig. 4.1(a), tree kernel needs to consult to the entire tree structure, which may be too sparse to be matched in the training data. By contrast, the subtree sequence based kernels can capture the pattern by a non-contiguous subtree sequence which consists of three subtrees in Fig. 4.1(f).

Third, the subtree sequence based kernels may reduce the negative effects brought by parsing errors in higher layers of parse trees. For common bottom up parsing algorithms, e.g. CKY algorithm (Hopcroft and Ullman, 1979), parsing errors tend to be more severe for larger structures with more height. To alleviate this problem, for an arbitrary span, the subtree sequence based kernels can match the lower layer subtree sequence structures while ignoring the higher structures. For example, when enumerating the structures in Fig. 4.1(b), the structure over the context "*the fast crackdown on smuggling*" (in dash line) cannot be captured by tree kernels due to parsing errors. Compared with the correct parse tree in Fig. 4.1(a), most structures over span $[2, 6]$ in Fig. 4.1(b) are correctly obtained except the rule covering the span $[2, 6]$. Alternatively, the subtree sequence based kernels can capture the correctly parsed structures (in dash line) using multiple subtrees. This may facilitate the matching and recovering of the correct structures corresponding to Fig. 4.1(a) (in dash line).

The tree sequence structure has been found to be effective in syntactic intensive applications, such as syntax based machine translation. Sun, Zhang, and Tan (2009) proposed the non-contiguous tree sequence based translation model which employs tree sequence as translation equivalences. In this model, the tree sequence structure

relaxes the structural constraint caused by the single tree based models without sacrificing the grammatical information embedded in each subtree. Hence, the tree sequence structure is beneficial in capturing the structural divergence beyond the syntactic constraints across languages. Witnessing their success, we attempt to apply the tree sequence structure to construct new kernels on parse trees.

4.2 Contiguous Tree Sequence Kernel

In this section, we propose contiguous Tree Sequence Kernel (cTSK) to explore the feature space of a sequence of subtrees in the contiguous context. We first attempt to evaluate the kernel by adapting the algorithm of Collins and Duffy's tree kernel, which is accomplished by modifying the tree structures. Alternatively, we propose a more efficient algorithm achieving the same time complexity with Collins and Duffy's tree kernel.

4.2.1 Kernel Evaluation via Pseudo Roots

Inspired by Collins and Duffy's tree kernel, we initially attempt to evaluate cTSK using the same idea. To inherit the algorithm of tree kernel, we transform a subtree sequence structure to a *pseudo root directed subtree*. As shown in Fig 4.2 [1] , we first create a *universally identical* pseudo root node for each subtree sequence denoted as a solid dot. Then edges are created to connect the pseudo root to the root of each subtree. This helps to transform a subtree sequence to a single tree structure. Since all pseudo roots are identical, it guarantees the *pseudo root directed subtrees* to be matched from the beginning at the pseudo roots. Based on the algorithm of tree

[1] In fact, the pseudo nodes are created for subtree sequence with arbitrary number of subtrees. However, we only illustrate the tree sequence with subtree number larger than 1 for clear illustration. Obviously, the structure over one single subtree can be constructed by linking the pseudo root to the root of the subtree.

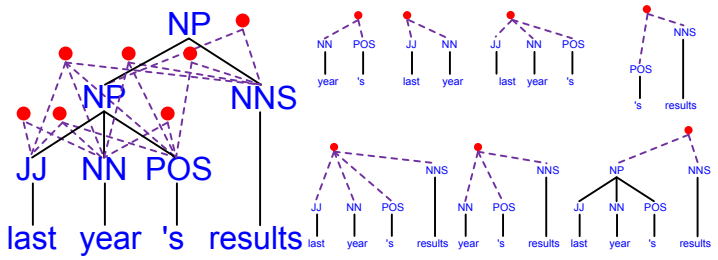


Figure 4.2: Illustration of Pseudo Root construction

kernel in Eq. 3.22, which iterates over all combinations of nodes in the given two parse trees, the cTSK function $K_{cts}(T, T')$ can be similarly evaluated by iterating over all pairs of pseudo roots as follows:

$$\begin{aligned} K_{cts}(T, T') &= \sum_{j=1}^{|\mathcal{T}_{prds}|} \left(\sum_{p \in P_T} I_j(p) \cdot \sum_{p' \in P_{T'}} I_j(p') \right) \\ &= \sum_{p \in P_T} \sum_{p' \in P_{T'}} \Psi(p, p') \end{aligned} \tag{4.1}$$

where P_T and $P_{T'}$ refer to the pseudo root sets of T and T' respectively. $\mathcal{T}_{prds}$ refers to the original feature space of *pseudo root directed subtrees*. To evaluate $\Psi(p, p')$, we define a matching function $\varphi(p, p') \in \{0, 1\}$ for each pair of pseudo roots as a prerequisite for structure matching. In the matching function, $p.child\#$ refers to the number of the child nodes of the pseudo root p, which is also the number of subtrees in the original subtree sequence. Therefore, $\varphi(p, p')$ equals to 1 if p and p' have identical number of child nodes, otherwise 0.

$$\varphi(p, p') = \begin{cases} 1 & \text{if } p.child\# = p'.child\# \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Then $\Psi(p, p')$ can be calculated as:

$$\Psi(p, p') = \mu^{p.child\#} \cdot \varphi(p, p') \cdot \Lambda(p, p') \tag{4.3}$$

where $\mu \in [0, 1]$ is a decay factor for the number of subtrees in a given tree sequence. Consequently, cTSK can be evaluated over all pairs of pseudo roots p and p', by means of $\Lambda(p, p')$ defined in Eq. 3.22.

Compared with tree kernel, the number of node pairs traversed is enlarged from the original tree node sets to the pseudo root sets. Therefore, the computational complexity is $O(|P_T| \cdot |P_{T'}|)$. Although we can achieve the kernel evaluation in polynomial time with respect to the number of pseudo roots, the actual computational cost may be high. The construction of the pseudo roots requires enumerating all the tree node sets that cover a contiguous span. Hence, the worst time cost for this algorithm could be exponential with respective to the sentence length.

4.2.2 Algorithm 2: Fast Evaluation

Alternatively, we propose an efficient algorithm to evaluate cTSK. The general goal of this algorithm is to match the subtrees in a subtree sequence from left to right. The key issue to achieve this goal is the production rules rooted at each subtree. Suppose the production rule at the root node N of a subtree is matched. On the one hand, we can proceed vertically from the child nodes of N to match the subtrees in the lower layers. This can be achieved by the tree kernel function $\Lambda(n, n')$ in Eq. 3.22. On the other hand, we can horizontally move to the next subtree which is adjacent to the subtree rooted at N. Therefore, the subtree sequence matching problem can be simplified into a subtree matching problem. Since the kernel value of the matched subtrees can be reused, it is straightforward to perform the algorithm by dynamic programming as follows:

Initially, we define a matching function $\Omega(n, n') \in \{0, 1\}$ to compare the production rules whose left hand side nonterminals are $n \in N_T$ and $n' \in N_{T'}$ respec-

tively. $\Omega(n, n')$ returns 1 if the productions are identical, otherwise 0.

$$\Omega(n, n') = \begin{cases} 1 & \text{if } n.production = n'.production \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

We further define $\Delta(i, j)$ to evaluate all the matched subtree sequences that ended at indices i of T and j of T'.

$$\Delta(i,j) = \begin{cases} \sum_{\substack{n \in N_T,\ n_e = i \\ n' \in N_{T'},\ n'_e = j}} \binom{\mu \cdot \Omega(n,n')\Lambda(n,n') \cdot}{(1+\Delta(n_b-1, n'_b-1))} & \text{if } n_b \geq 1, n'_b \geq 1 \\ \sum_{\substack{n \in N_T,\ n_e = i \\ n' \in N_{T'},\ n'_e = j}} \mu \cdot \Omega(n,n')\Lambda(n,n') & \text{otherwise} \end{cases} \tag{4.5}$$

where l and l' refer to the number of leaf nodes of T and T'. Therefore, we have $0 \leq i < l,\ 0 \leq j < l'$. We explain the recursive function $\Delta(i, j)$ as follows:

(1) In each evaluation of $\Delta(i, j)$, it traverses all the node pairs (n, n') with $n_e = i$ and $n'_e = j$. For any node n, n_b refers to the beginning index of the span covered by n, while n_e refers to the end index of the span.

(2) When evaluating the node pair (n, n'), $\Omega(n, n')$ is firstly evaluated. If $\Omega(n, n')$ equals to 1, which means the production rules rooted at n and n' are identical, it will continue to measure the structure similarity by computing the tree kernel function $\Lambda(n, n')$. As defined in Eq. 3.22, $\Lambda(n, n')$ returns the number of common subtrees rooted at (n, n').

(3) The value $\Lambda(n, n')$ is then utilized in two folds. For the case that the subtree sequence consists of one single subtree, the value is just $\Lambda(n, n')$. For the case that the subtree sequence consists of multiple subtrees, with the subtrees rooted at (n, n') to be the last subtree in the sequence, the value is conveyed by $\Lambda(n, n')\Delta(n_b - 1, n'_b - 1)$.

To achieve the complete kernel evaluation of $K_{cts}(T, T')$, we sum up all index combinations of (i, j) for $\Delta(i, j)$ as follows:

$$K_{cts}(T, T') = \sum_{\substack{0 \leq i < l \\ 0 \leq j < l'}} \Delta(i, j) \tag{4.6}$$

Inspired by tree kernel, which employs λ to penalize structures with much height, we introduce another decay factor μ to penalize the number of subtrees in a subtree sequence. The factor μ is incurred at each time a subtree is matched. By means of μ, a subtree sequence consisting of multiple subtrees is discounted.

As shown above, the evaluation of $\Delta(i, j)$ can be accomplished by dynamic programming. For the ease of understanding, we analyze the time complexity in a naive way. We can evaluate the tree kernel function $K_t(T, T')$ before the evaluation of $K_{cts}(T, T')$ to ensure a complete realization of $\Lambda(.,.)$, which costs $O(|N_T| \cdot |N_{T'}|)$. Hence, when we evaluate the matching function $\Delta(i, j)$ for $K_{cts}(T, T')$, the tree kernel function $\Lambda(.,.)$ is already available and unnecessary to compute. Note that although the evaluation of all $\Delta(i, j)$, $0 \leq i < l,\ 0 \leq j < l'$ traverses all indices (i, j), in fact, all (n, n') for each (i, j) with $n_e = i$ and $n'_e = j$ will be traversed. Therefore, the time complexity incurred by the evaluation in this step is $O(|N_T| \cdot |N_{T'}|)$ instead of $O(|l| \cdot |l'|)$. Finally, the sum of all $\Delta(i, j)$ for $K_{cts}(T, T')$ needs to traverse all indices (i, j), costing $O(|l| \cdot |l'|)$. Consequently, the overall time complexity is incurred sequentially by the three elements, accomplished in $O(|N_T| \cdot |N_{T'}| + |N_T| \cdot |N_{T'}| + |l| \cdot |l'|)$. For normal labeled ordered rooted trees, $N_T > l$. Therefore, the final computational complexity is $O(|N_T| \cdot |N_{T'}|)$, the same with Collins and Duffy's tree kernel.

In real implementation, the matching functions $\Lambda(.,.)$ and $\Delta(.,.)$ should be evaluated simultaneously, so that the computation of $\Lambda(.,.)$ will be incurred only when it is necessary. The sum of all (i, j) for $K_{cts}(T, T')$ can be performed with the incremental evaluation of $\Delta(i, j)$.

4.3 Set Sequence Kernels

From the last section, we can find that cTSK can be evaluated by simultaneously performing the matching of subtree root nodes from left to right and the matching

of the structures covered by a root node from top to bottom. The latter matching problem can be achieved by using the $\Lambda(.,.)$ function defined in Collins and Duffy's tree kernel (Eq. 3.22). The former matching problem is actually not a tree structure matching problem but a sequence matching problem. In terms of cTSK, the sequence of subtree roots covering a contiguous span is matched across the tree pair. In order to generalize the problem of matching the sequence of subtree root nodes, we extend the sequence kernel (Section 3.1.3) to propose Set Sequence Kernel (SSK), which allows multiple choices of node symbols in any position of a sequence. SSKs are then combined with tree kernel to accomplish the evaluation of TSKs in Section 4.4, which allow the subtree sequence to be non-contiguous.

SSK is defined in the same style with sequence kernel in Section 3.1.3. Let Σ be a finite symbol set. Let $S = S_0 S_1 \cdots S_{|S|-1}$ be a *Set Sequence* with each item S_i to be a set, where each element[2] in the set is uniquely indexed from 0 to $|S_i|-1$ and is labeled with a symbol $s \in \Sigma$. Let $s(i, \hat{i}) \in S_i, 0 \leq \hat{i} < |S_i|$ be the $\hat{i}$-th element in set S_i. Let $(\mathbf{i}, \hat{\mathbf{i}}) = [(i_1, \hat{i}_1), (i_2, \hat{i}_2), \ldots, (i_m, \hat{i}_m)]$, where $0 \leq i_1 < i_2 < \cdots < i_m \leq |S| - 1$ and $0 \leq \hat{i}_k < |S_k|$, be a subset of index pairs in S. In each index pair, the first element denotes the index of the set in a given Set Sequence and the second element denotes a specific element in this set. Then let $s[(\mathbf{i}, \hat{\mathbf{i}})]$ refer to a sequence of elements $s(i_1, \hat{i}_1)s(i_2, \hat{i}_2) \cdots s(i_m, \hat{i}_m)$.

For all sequences of elements $u = s[(\mathbf{i}, \hat{\mathbf{i}})]$ and $u' = s'[(\mathbf{i}, \hat{\mathbf{i}})]$, we write $u \doteq u'$ in case the number of elements in u and u' are identical and the corresponding symbols in the same position are identical. We say u and u' are matched in this case. In addition, for a sequence of symbols w, we write $u \doteq w$ to refer to the case that the symbol sequence labeled for u is w.[3]

[2] An element is uniquely identified by the index pair $(i, \hat{i})$. The symbol labeled on the element is only considered as a feature for this element.

[3] u is a sequence of elements, while w is a sequence of symbols. Since each element is labeled by a symbol, an element sequence can be considered to be labeled by a symbol sequence.

Finally, let $\widetilde{K}_m(S, S')$ be the number of matched sequences of elements for S and S', with the number of elements in the sequence no more than m. Then for all m, the kernel is defined as

$$\widetilde{K}_m(S, S') = \sum_{w=\Sigma^m} \left(\sum_{\substack{(\mathbf{i},\hat{\mathbf{i}}): \\ u=s[(\mathbf{i},\hat{\mathbf{i}})] \\ u \doteq w}} p(u, \mathbf{i}) \cdot \sum_{\substack{(\mathbf{j},\hat{\mathbf{j}}): \\ u'=s'[(\mathbf{j},\hat{\mathbf{j}})] \\ u' \doteq w}} p(u', \mathbf{j}) \right) \tag{4.7}$$

According to sequence kernel, the penalizing function $p(u, \mathbf{i})$ can be the lexical span length. Although this may work well for character based sequence with each item covering exactly length of 1, it is very likely that the method would not adapt well on tree sequence, since the size of subtrees varies a lot. In addition, penalty on spans may violate the purpose of matching syntactic tree structures, that tree structures covering different length of spans are identically considered. As a result, instead of adapting the standard sequence kernel function, we propose new kernel functions for *Set Sequence* using two alternative penalizing approaches. One is to penalize the *count of matched elements*. The other is to penalize the *number of gaps* ignoring the span length i.e. each gap between two matched elements only incurs the decay factor once, no matter how large span the gap covers.

To facilitate the illustration of TSK in later sections, we define $K_m(i, j)$ to be the kernel value that matches all the subsequences up to i and j with length no more than m. Therefore, SSK that matches all the subsequences with length no more than m can be defined as $K_m(|S| - 1, |S'| - 1) = \widetilde{K}_m(S, S')$, which evaluates elements from the *0*-th set to the *last* set of S and S'. Hence we transform the kernel function from $\widetilde{K}_m(S, S')$ to $K_m(i, j)$, where $0 \leq i < |S|$ and $0 \leq j < |S'|$ and evaluate $K_m(i, j)$ using dynamic programming.

Later, we propose SSKs with different structure weighting schemes. These weighting schemes are based on three basic approaches to penalize structures. γ is

used to penalize the length of spans, which is similar to sequence kernels. μ is used to penalize the count of matched elements. τ is used to penalize the count of gaps.

4.3.1 Penalizing Length of Spans (γ)

The standard sequence kernel (Lodhi et al., 2002; Cancedda et al., 2003) devises the weighting function Eq. 3.19 to penalize the subsequence. The number of *matched elements* and the *length of gaps* are penalized using the same decay factor. In other words, it penalizes the length of span covered by the given matched subsequences. By adapting from sequence kernel, the kernel function $K_m(i,j)$, which penalizes the length of spans can be evaluated as follows.

$$
\begin{aligned}
K_m^{(l)}(i,j) &= 0 \qquad \text{if } \min(i,j) < m-1 \text{ or } i,j < 0 \quad \text{for all } l \in \{1,2,3\} \\
K_m(i,j) &= 0 \qquad \text{if } \min(i,j) < m-1 \text{ or } i,j < 0 \\
K_m^{(2)}(i,j) &= 1 \qquad \text{if } m = 0 \\
K_m^{(3)}(i,j) &= \gamma K_m^{(3)}(i,j-1) + \sum_{s_i \in S_i} \sum_{\substack{s'_j \in S'_j \\ s_i \doteq s'_j}} \left(\gamma^2 K_{m-1}^{(2)}(i-1,j-1) \right) \qquad (4.8) \\
K_m^{(2)}(i,j) &= \gamma K_m^{(2)}(i-1,j) + K_m^{(3)}(i,j) \qquad (4.9) \\
K_m^{(1)}(i,j) &= K_m^{(1)}(i,j-1) + \sum_{s_i \in S_i} \sum_{\substack{s'_j \in S'_j \\ s_i \doteq s'_j}} \left(\gamma^2 K_{m-1}^{(2)}(i-1,j-1) \right) \qquad (4.10) \\
K_m(i,j) &= K_m(i-1,j) + K_m^{(1)}(i,j) \qquad (4.11)
\end{aligned}
$$

In order to evaluate $K_m(i,j)$, we introduce some intermediate matching functions i.e. $K_m^{(1)}(i,j)$, $K_m^{(2)}(i,j)$ and $K_m^{(3)}(i,j)$. In Eq. 4.11, $K_m(i,j)$ is evaluated by summing up the kernel value which does not match the elements in the i-th set S_i, i.e. $K_m(i-1,j)$ and the kernel value which matches the elements in the i-th set S_i, i.e. $K_m^{(1)}(i,j)$. In Eq. 4.10, given that the i-th set S_i is matched, $K_m^{(1)}(i,j)$ is evaluated by summing up the kernel value which does not match the elements in the j-th

set S'_j, i.e. $K_m^{(1)}(i, j-1)$ and the kernel value which matches the elements in the i-th set S_i and the elements in the j-th set S'_j, i.e. $\sum_{s_i \in S_i} \sum_{\substack{s'_j \in S'_j \\ s_i \doteq s'_j}} \left(\gamma^2 K_{m-1}^{(2)}(i-1, j-1)\right)$.

Let $K_m^{(2)}(i, j)$ count the number of matches of sequences discounted by γ from the set indices of the first matched elements to the indices of i of S and j of S' respectively, no matter whether the elements in the i-th set S_i and j-th set S'_j are matched. Let $K_m^{(3)}(i, j)$ count the number of matches of subsequences discounted by γ from the set indices of the first matched elements to the indices of i of S and j of S' respectively, which requires the element in the i-th set S_i to be matched. Therefore, in Eq. 4.9, $K_m^{(2)}(i, j)$ can be evaluated by summing up the kernel value which does not match the elements in the i-th set S_i, i.e. $\gamma K_m^{(2)}(i-1, j)$ and the kernel value which matches the elements in the i-th set S_i, i.e. $K_m^{(3)}(i, j)$. Similarly, in Eq. 4.8, given that the i-th set S_i is matched, $K_m^{(3)}(i, j)$ is evaluated by summing up the kernel value which does not match the elements in the j-th set S'_j, i.e. $\gamma K_m^{(3)}(i, j-1)$ and the kernel value which matches the elements in the i-th set S_i and the elements in the j-th set S'_j, i.e. $\sum_{s_i \in S_i} \sum_{\substack{s'_j \in S'_j \\ s_i \doteq s'_j}} \left(\gamma^2 K_{m-1}^{(2)}(i-1, j-1)\right)$.

4.3.2 Penalizing Count of Matched Elements (μ)

In this section, we propose the kernel function $K_m(i, j)$, which penalizes the count of matched elements as follows.

$$
\begin{aligned}
K_m^{(l)}(i, j) &= 0 \qquad \text{if } \min(i, j) < m-1 \text{ or } i, j < 0 \text{ for all } l \in \{1, 2\} \\
K_m^{(1)}(i, j) &= 1 \qquad \text{if } m = 0 \\
K_m^{(2)}(i, j) &= K_m^{(2)}(i, j-1) + \sum_{s_i \in S_i} \sum_{\substack{s'_j \in S'_j \\ s_i \doteq s'_j}} K_{m-1}^{(1)}(i-1, j-1) \qquad (4.12) \\
K_m^{(1)}(i, j) &= K_m^{(1)}(i-1, j) + K_m^{(2)}(i, j) \qquad (4.13) \\
K_m(i, j) &= \mu^m \cdot K_m^{(1)}(i, j) \qquad (4.14)
\end{aligned}
$$

To evaluate $K_m(i,j)$, we use two intermediate matching functions $K_m^{(1)}(i,j)$ and $K_m^{(2)}(i,j)$. Let $K_m^{(1)}(i,j)$ refer to the kernel values when the penalty factor μ is not taken into account. In other words, each pair of matched subsequence contributes exactly 1 to the kernel $K_m^{(1)}(i,j)$. In Eq. 4.13, $K_m^{(1)}(i,j)$ is evaluated by the kernel value which does not match the elements in the i-th set S_i, i.e. $K_m^{(1)}(i-1,j)$ and the value which matches the elements in the i-th set S_i, i.e. $K_m^{(2)}(i,j)$. In Eq. 4.12, given that the i-th set S_i is matched, $K_m^{(2)}(i,j)$ is evaluated by the kernel value which does not match the elements in the j-th set S'_j, i.e. $K_m^{(2)}(i,j-1)$ and the kernel value which matches elements in the i-th set S_i and the elements in the j-th set S'_j, i.e. $\sum_{s_i \in S_i} \sum_{\substack{s'_j \in S'_j \\ s_i \doteq s'_j}} K_{m-1}^{(1)}(i-1,j-1)$. Finally, $K_m(i,j)$ is evaluated by $K_m^{(1)}(i,j)$ multiplied by the penalty to the power of the number of elements allowed in the matched subsequences in Eq. 4.14.

4.3.3 Penalizing Count of Gaps (τ)

The kernel to penalize the count of gaps in the sequence can be evaluated as follows.

$$
\begin{aligned}
K_m^{(l)}(i,j) &= 0 \qquad \text{if } \min(i,j) < m-1 \text{ or } i,j < 0 \text{ for all } l \in \{1,2,3,4\} \\
K_m^{(l)}(i,j) &= 0 \qquad \text{if } m = 0 \text{ for all } l \in \{1,2,3\} \\
K_m^{(4)}(i,j) &= 1 \qquad \text{if } m = 0 \\
K_m^{(1)}(i,j) &= K_m(i-1,j-1) && (4.15) \\
K_m^{(2)}(i,j) &= K_m^{(2)}(i,j-1) + K_m^{(4)}(i,j-1) && (4.16) \\
K_m^{(3)}(i,j) &= K_m^{(3)}(i-1,j) + K_m^{(4)}(i-1,j) && (4.17) \\
K_m^{(4)}(i,j) &= \sum_{s_i \in S_i} \sum_{\substack{s'_j \in S'_j \\ s_i \doteq s'_j}} \begin{pmatrix} \tau^2 K_{m-1}^{(1)}(i-1,j-1) + \tau K_{m-1}^{(2)}(i-1,j-1) \\ +\tau K_{m-1}^{(3)}(i-1,j-1) + K_{m-1}^{(4)}(i-1,j-1) \end{pmatrix} && (4.18) \\
K_m(i,j) &= \sum_{l=1}^{4} K_m^{(l)}(i,j) && (4.19)
\end{aligned}
$$

Similar to the previous two SSKs, we also propose certain intermediate matching functions, i.e. $K_m^{(l)}(i,j),\ l \in \{1,2,3,4\}$, to accomplish efficient evaluation of $K_m(i,j)$. Let $K_m^{(1)}(i,j)$ refer to the case where neither the elements in set S_i or the elements in set S'_j are matched in the subsequence. Let $K_m^{(2)}(i,j)$ refer to the case where an element in set S_i is matched but none of the elements in set S'_j are matched in the subsequence. Let $K_m^{(3)}(i,j)$ refer to the case where none of the elements in set S_i are matched but an element in set S'_j is matched in the subsequence. Let $K_m^{(4)}(i,j)$ refer to the case where an element in set S_i and an element in set S'_j match to each other in the subsequence. In Eq. 4.18, the decay factor τ is incurred in computing $K_m^{(4)}(i,j)$, when both elements are matched. The evaluation of $K_m(i,j)$ can be achieved by the sum of all the four cases.

Generally, the evaluation of all the three SSKs can be accomplished in time complexity of $O(m|\Sigma_{i=0}^{|S|}S_i| \cdot |\Sigma_{j=0}^{|S'|}S'_j|)$, which refers to the product of the element numbers in the two node sets.

4.4 Tree Sequence Kernels

In this section, we propose Tree Sequence Kernels (TSKs). Leveraging SSK and Collins and Duffy's tree kernel, we first present the generalized form for TSK. In addition, we verify that the generalized TSK is a valid kernel by showing that TSK can be constructed by the mapping kernel paradigm. After that three instantiations of TSK are proposed with different structure weighting schemes, corresponding to SSKs.

4.4.1 The Generalized Tree Sequence Kernel

We start this section with the generalized Tree Sequence Kernel. Let T refer to a tree with span length l. Let Σ refer to a finite label set. For all nodes $n \in N_T$,

where N_T refers to the node set consisting of all the nodes in T, n is labeled with $s \in \Sigma$. Let $N_0, N_1, \ldots, N_{|l-1|}$ refer to l ordered node sets. The node set N_i for all the indices $0 \leq i < l$ is constructed by delivering all the nodes $n \in N_T$ to set N_i, if and only if $n \in N_{|n_e|}$, where n_e refers to the end index of the span covered by n.

Based on the definition in Section 4.3, the object of $N_0 N_1 \cdots N_{|l-1|}$ can be considered as a *Set Sequence*. Therefore, let $n(i, \hat{i})$ be a node, i.e. $n(i, \hat{i}) \in N_i, 0 \leq \hat{i} < |N_i|$. Let $(\mathbf{i}, \hat{\mathbf{i}}) = [(i_1, \hat{i}_1), (i_2, \hat{i}_2), \ldots, (i_m, \hat{i}_m)]$, where $0 \leq i_1 < i_2 < \cdots < i_m \leq l-1$ and $0 \leq \hat{i}_k < |N_k|$, be a subset of index pairs. In each index pair, the first element denotes the index of a node set and the second element denotes a specific node in this node set. Let $n[(\mathbf{i}, \hat{\mathbf{i}})]$ refer to a sequence of nodes $n(i_1, \hat{i}_1) n(i_2, \hat{i}_2) \cdots n(i_m, \hat{i}_m)$, where for every adjacent node pair $n(i_p, \hat{i}_p)$ and $n(i_{p+1}, \hat{i}_{p+1})$ in the node sequence with $1 \leq p < m$, the *non-overlap* condition holds (Section 2.3). It's easy to see that if adjacent nodes in this node sequence satisfy the *non-overlap* condition, all the pair of nodes in this node sequence satisfy the condition.

Finally, let $\widetilde{K}_m(T, T')$ be the number of common subtree sequence structures for T and T', where the number of subtrees in a matched subtree sequence is no more than m. Let u refer to the *non-overlapped* root node sequence with each node sequentially corresponding to a subtree in a subtree sequence. Additionally, let $u[k]$ denote the kth node in u. Therefore, based on the $\Lambda(.,.)$ function defined in Eq. 3.22, the generalized TSK is

$$\widetilde{K}_m(T, T') = \sum_{w \in \Sigma^m} \left(\sum_{\substack{(\mathbf{i}, \hat{\mathbf{i}}): \\ u = n[(\mathbf{i}, \hat{\mathbf{i}})] \\ u \doteq w}} \sum_{\substack{(\mathbf{j}, \hat{\mathbf{j}}): \\ u' = n'[(\mathbf{j}, \hat{\mathbf{j}})] \\ u' \doteq w}} \prod_{k=1}^{m} \Lambda(u[k], u'[k]) \right) \tag{4.20}$$

Next we verify the positive semidefinite property of the above kernel by constructing it through the mapping kernel paradigm. It is clear from the closure of

kernels under tensor product (Eq. 2.28) that

$$\widetilde{k}_m(u, u') = \prod_{k=1}^{m} \Lambda(u[k], u'[k]) \tag{4.21}$$

is a kernel on $\Sigma^m \times \Sigma^m$. Therefore, the general tree sequence kernel can be constructed through the mapping kernel paradigm with $\widetilde{k}_m(u, u')$ to be the underlying kernel. Consequently, the generalized TSK can be rewritten in the form that

$$\widetilde{K}_m(T, T') = \sum_{(u,u') \in \mathcal{M}_{T,T'}} \widetilde{k}_m(u, u') \tag{4.22}$$

where $\mathcal{M}_{T,T'}$ is a finite and symmetric set satisfying that

$$\mathbb{M} = (\{\mathcal{T} | T, T' \in \mathcal{T}\}, \mathcal{U}_T^m, \{\mathcal{M}_{T,T'} \subseteq \mathcal{U}_T^m \times \mathcal{U}_{T'}^m | \forall (u, u') \in \mathcal{M}_{T,T'} \ , \ u \doteq u'\}) \tag{4.23}$$

where $\mathcal{U}_T^m$ refers to the set of *non-overlapped* node sequences with m nodes in T.

It is easy to verify that $\mathbb{M}$ is transitive under the $u \doteq u'$ constraint. As a result, the generalized TSK is a valid positive semidefinite kernel.

4.4.2 Adapting Set Sequence Kernel to Tree Sequence Kernel

In order to evaluate TSK, we integrate the $\Lambda(.,.)$ function in Collins and Duffy's tree kernel into the algorithms of SSK by means of certain modifications.

To apply the SSK algorithm, we first transform the parse tree structure into a valid Set Sequence structure. In Fig. 4.3, we construct the Set Sequence for a parse tree with span $[0, 4]$ by sending the internal nodes to the set $N_0, \ldots, N_4$. For example, since NN^1 is ended with index 1, we put NN^1 in N_1.

The general idea of the TSK evaluation is to match the subtrees in a subtree sequence from left to right and from top to bottom. Given a subtree sequence to be matched, the key issue is to match the root node of each subtree. When the root node n of a subtree t is matched, we need to *vertically* move downwards and

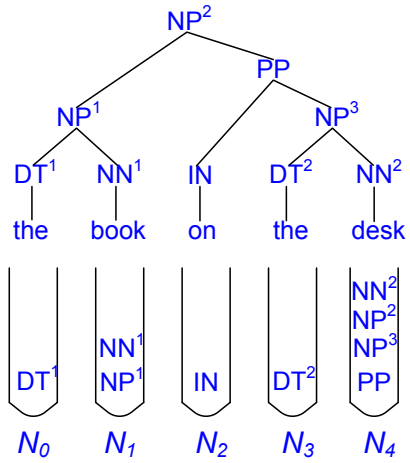


Figure 4.3: Construction of Node Set Sequence

match the entire subtree structure t. In addition, we also need to *horizontally* move to the right side of t and match the root node n' of the subtree t' that is adjacent to t. A gap is allowed between the subtree t and the subtree t'. Consequently, the subtree sequence matching problem is transformed into a problem of matching the root node sequences and the single tree structures. To implement this idea in dynamic programming, the *horizontal* evaluation is achieved by incurring SSK on the node sequence set of the given parse tree pair. In other words, any matched node in the node set will be considered as the root of the subtree to be matched. At the same time, the *vertical* evaluation is achieved by using the tree kernel function $\Lambda(.,.)$ within the *root-matched* subtrees.

Before presenting the algorithms, we define certain notations. For any node n let $l(n) = n_e - n_b + 1$, where n_b and n_e refer to the beginning and end index of the span covered by n. The evaluation of TSKs require certain modifications of SSKs. Briefly, the adaptation of SSKs to TSKs is achieved by three major modifications:

First, TSKs incur the tree kernel computation of $\Lambda(n, n')$ when the nodes n and n' are matched and the conditions of $n_e = i$ and $n'_e = j$ are satisfied.

Second, when the nodes n and n' are matched, it reuses the kernel values before

the indices of (n_b, n'_b). This avoids the matching of other nodes in the span covered by (n, n'), since any two subtrees in a given subtree sequence do not *overlap*. This requires changing the indices of $(i-1, j-1)$ to $(n_b - 1, n'_b - 1)$.

Third, for the factor of γ to penalize the count of matched subtrees, the power of γ incurred when matching a node pair is $(l(n) + l(n'))$ instead of 2.

Therefore, with other equations being the same, we only need to modify certain kernel functions of SSKs to achieve the evaluation of TSKs.

4.4.3 Penalizing Length of Spans (γ)

According to the three modification schemes, TSK with penalty factor γ can be evaluated by modifying Eq. 4.10 into Eq. 4.26 and Eq. 4.8 into Eq. 4.24.

$$
\begin{aligned}
K_m^{(l)}(i,j) &= 0 \qquad \text{if } \min(i,j) < m-1 \text{ or } i,j < 0 \text{ for all } l \in \{1,2,3\} \\
K_m(i,j) &= 0 \qquad \text{if } \min(i,j) < m-1 \text{ or } i,j < 0 \\
K_m^{(2)}(i,j) &= 1 \qquad \text{if } m = 0 \\
K_m^{(3)}(i,j) &= \gamma K_m^{(3)}(i,j-1) + \sum_{n_e=i} \sum_{\substack{n'_e=j \\ n \doteq n'}} \left(\gamma^{l(n)+l(n')} K_{m-1}^{(2)}(n_b - 1, n'_b - 1) \Lambda(n,n') \right)
\end{aligned}
\tag{4.24}
$$

$$
K_m^{(2)}(i,j) = \gamma K_m^{(2)}(i-1,j) + K_m^{(3)}(i,j) \tag{4.25}
$$

$$
K_m^{(1)}(i,j) = K_m^{(1)}(i,j-1) + \sum_{n_e=i} \sum_{\substack{n'_e=j \\ n \doteq n'}} \left(\gamma^{l(n)+l(n')} K_{m-1}^{(2)}(n_b - 1, n'_b - 1) \Lambda(n,n') \right) \tag{4.26}
$$

$$
K_m(i,j) = K_m(i-1,j) + K_m^{(1)}(i,j) \tag{4.27}
$$

4.4.4 Penalizing Count of Matched Subtrees (μ)

TSK with penalty factor μ can be evaluated by modifying Eq. 4.12 into Eq. 4.28.

$$K_m^{(l)}(i,j) = 0 \qquad \text{if } \min(i,j) < m-1 \text{ or } i,j < 0 \text{ for all } l \in \{1,2\}$$

$$K_m^{(1)}(i,j) = 1 \qquad \text{if } m = 0$$

$$K_m^{(2)}(i,j) = K_m^{(2)}(i,j-1) + \sum_{n_e=i} \sum_{\substack{n'_e=j \\ n \doteq n'}} K_{m-1}^{(1)}(n_b-1, n'_b-1) \cdot \Lambda(n,n') \tag{4.28}$$

$$K_m^{(1)}(i,j) = K_m^{(1)}(i-1,j) + K_m^{(2)}(i,j) \tag{4.29}$$

$$K_m(i,j) = \mu^m \cdot K_m^{(1)}(i,j) \tag{4.30}$$

4.4.5 Penalizing Count of Gaps (τ)

In addition, TSK with the factor τ to penalize the count of gaps can be evaluated by modifying Eq. 4.18 into Eq. 4.34.

$$K_m^{(l)}(i,j) = 0 \qquad \text{if } \min(i,j) < m-1 \text{ or } i,j < 0 \text{ for all } l \in \{1,2,3,4\}$$

$$K_m^{(l)}(i,j) = 0 \qquad \text{if } m = 0 \text{ for all } l \in \{1,2,3\}$$

$$K_m^{(4)}(i,j) = 1 \qquad \text{if } m = 0$$

$$K_m^{(1)}(i,j) = K_m(i-1,j-1) \tag{4.31}$$

$$K_m^{(2)}(i,j) = K_m^{(2)}(i,j-1) + K_m^{(4)}(i,j-1) \tag{4.32}$$

$$K_m^{(3)}(i,j) = K_m^{(3)}(i-1,j) + K_m^{(4)}(i-1,j) \tag{4.33}$$

$$K_m^{(4)}(i,j) = \sum_{n_e=i} \sum_{\substack{n'_e=j \\ n \doteq n'}} \begin{pmatrix} \tau^2 K_{m-1}^{(1)}(n_b-1, n'_b-1) \\ +\tau K_{m-1}^{(2)}(n_b-1, n'_b-1) \\ +\tau K_{m-1}^{(3)}(n_b-1, n'_b-1) \\ +K_{m-1}^{(4)}(n_b-1, n'_b-1) \end{pmatrix} \cdot \Lambda(n,n') \tag{4.34}$$

$$K_m(i,j) = \sum_{l=1}^{4} K_m^{(l)}(i,j) \tag{4.35}$$

In fact, if the minimum matched structure is restricted to a CFG rule with the height of 2, the set sequence can be constructed by the set of production rules,

instead of simple nodes. In real implementation, we also rank the elements by their labels in the alphabetical order for each node set (as shown in Fig. 4.3) to achieve fast node matching (Moschitti, 2006). The time complexity of TSKs is incurred by the time cost of SSKs and the computation of the $\Lambda(.,.)$ function. If we consider that tree kernel is evaluated before SSK to make the $\Lambda(.,.)$ function available before evaluating SSKs. The cost is the sum of both kernels to be $O(m|N_T| \cdot |N_{T'}| + |N_T| \cdot |N_{T'}|)$, where m is the maximum number of subtrees allowed to be matched. In fact, we evaluate TSKs by incurring SSKs and Collins and Duffy's tree kernel simultaneously, since partial results of both kernels can be reused. Briefly, the overall cost is $O(m|N_T| \cdot |N_{T'}|)$.

4.5 Anchored Tree Sequence Kernel

In NLP, many tasks deal with the instances with multiple relational target constituents, e.g. relation extraction, coreference resolution and semantic role analysis. The concerned text units that the corresponding tasks may aim to evaluate (e.g. entity pairs in relation extraction, NP and pronouns in coreference resolution, predicate and argument in semantic role analysis) can be recognized as the target constituents. Generally, there is a certain relationship among these target constituents. It is often required to analyze the role of the target constituents and identify the relationship among these constituents.

Leveraging TSKs, we present Anchored Tree Sequence Kernel (aTSK) to better facilitate these target constituents oriented tasks. aTSK identifies the target constituents using the anchored subtrees structures(Section 2.3) in parse trees. When exploring the tree sequence features in the parse trees with anchored subtrees, aTSK requires the structure features to be matched according to their relative positions to the anchored subtrees. In other words, structure features can be matched only if they are at the similar position relative to the anchored structures. This char-

acteristic may avoid the over-generation of noise features and may enhance the correspondence of the structure features to the target constituents. Besides aTSK, we also propose additional constraints to extend aTSK under the Mapping Kernel paradigm.

4.5.1 Feature Space Construction

We first design the feature space explored by aTSK. Let $t_1, t_2, \ldots, t_r$ and $t'_1, t'_2, \ldots, t'_r$ be the anchored subtrees for T and T' respectively.[4] Since aTSK aims to encapsulate the structure feature with its relative position to the anchored subtrees, we design the feature space explored by aTSK according to the following criterion. For all corresponding subtrees t in T and t' in T' in a matched subtree sequence structure,

a. for every $1 \leq k \leq r$, if t_k and t are *non-overlapped*, then t'_k and t' are *non-overlapped*;

b. for every $1 \leq k \leq r$, if t_k and t are *overlapped*, then t'_k and t' are *overlapped*, specifically

   (1) if the root of t is the ancestor of the root of t_k, then the root of t' is the ancestor of the root of t'_k;

   (2) if the root of t is the descendant of the root of t_k, then the root of t' is the descendant of the root of t'_k;

   (3) if the root of t is the root of t_k, then the root of t' is the root of t'_k.

Condition (a) indicates that the corresponding subtrees maintains the left/right position relative to the anchored subtree counterparts across the tree pair. Condition (b) indicates that the corresponding subtrees maintains the hierarchical po-

[4] the number of anchored subtrees in the parse tree is restricted to be the same across all parse trees

sition relative to the anchored subtree counterparts. In addition, it's not allowed that t is *overlapped* with t_k, while t' is *non-overlapped* with t'_k.

In order to realize the above constraints, trees are partitioned into multiple parts and each part is individually matched with the corresponding part in the other parse tree. According to Eq. 4.20, the essential issue of matching the subtree sequence structures is to match the *non-overlapped* node sequences. In Section 4.4.2, the generalized TSK is adapted to SSK by delivering each node of the original tree to one of the node sets $N_0, N_1, \ldots, N_{|l-1|}$. The node set each node is delivered to is identified according to the end index of span covered by the node. As a result, the task of partitioning the tree structure can be transfered to the problem of partitioning the node sets $N_0, N_1, \ldots, N_{|l-1|}$. In consequence, the partition of the original tree structure can be further addressed by partitioning the span $[0, l-1]$ of the tree.

Thus, given that tree T with anchored subtrees $t_1, t_2, \ldots, t_r$ covers the span $[0, l-1]$, we define **Span Partition** of T, denoted as $\vec{\vartheta}(T, \{t_1, t_2, \ldots, t_r\})$. $\vec{\vartheta}$ is a vector of subspans, where $\vec{\vartheta}(T, \{t_1, t_2, \ldots, t_r\}) = ([0, t_{1_b}), [t_{1_b}, t_{1_e}], [t_{1_e}+1, t_{2_b}), \ldots, [t_{r_e}+1, l-1])^T$, with $2r+1$ dimensions.

$\vec{\vartheta}$ can be constructed as follows:

- For each anchored subtree t_k, $1 \leq k \leq r$, the span $[t_{k_b}, t_{k_e}]$ forms a non-empty subspan;
- For each pair of adjacent anchored subtrees t_k and t_{k+1}, $1 \leq k < r$, the span $[t_{k_e}+1, t_{k+1_b})$ forms a subspan, which is possible to be empty;
- The span $[0, t_{1_b})$ forms a subspan, which is possible to be empty;
- The span $[t_{r_e}+1, l-1]$ forms a subspan, which is possible to be empty.

With respect to $\vec{\vartheta}(T, \{t_1, t_2, \ldots, t_r\})$, we define **Node Set Partition** for T with anchored subtrees $t_1, t_2, \ldots, t_r$, denoted as $\vec{\varrho}(T, \{t_1, t_2, \ldots, t_r\})$. The Partition

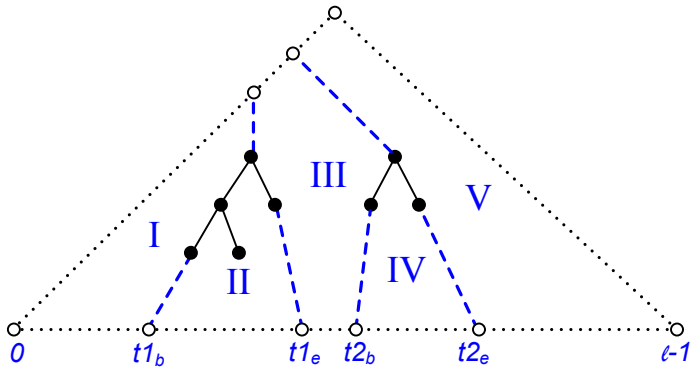


Figure 4.4: Tree Structure Partition.

$\vec{\varrho}(T, \{t_1, t_2, \ldots, t_r\})$ is:

- A $2r+1$ dimensional vector of disjoint subsets of N_T whose union is N_T;
- $\varrho_i = \{n | \forall k, \vartheta_{i_b} \leq k \leq \vartheta_{i_e}, n \in N_k\}$, for $0 \leq k \leq l-1$;

where ϑ_i refers to the i-th subspan of $\vec{\vartheta}$, which starts at index ϑ_{i_b} and ends at ϑ_{i_e}.

From the second condition, it is easy to find that $\vec{\varrho}(T, \{t_1, t_2, \ldots, t_r\})$ can be constructed by partitioning the node sets of $N_0, N_1, \ldots, N_{|l-1|}$ of $\vec{N}$ into $2r+1$ sets of node sets. For example in Fig. 4.4, the nodes in T are partitioned into five subsets. The nodes in the sets $N_0, \ldots, N_{t_{1_b}-1}$ forms the first subset ϱ_0. The nodes in the sets $N_{t_{1_b}}, \ldots, N_{t_{1_e}}$ forms the second subset ϱ_1. The nodes in the sets $N_{t_{1_e}+1}, \ldots, N_{t_{2_b}-1}$ forms the third subset ϱ_2. The nodes in the sets $N_{t_{2_b}}, \ldots, N_{t_{2_e}}$ forms the fourth subset ϱ_3. The nodes in the sets $N_{t_{2_e}+1}, \ldots, N_{l-1}$ forms the fifth subset ϱ_4.

All TSKs proposed in Section 4.4 can be extended to corresponding aTSKs by evaluating the kernels on each node subset obtained from the partition ϱ. Therefore, the algorithms in Section 4.4 can be modified respectively to achieve the evaluation of corresponding aTSKs.

Before proposing aTSKs, we introduce some notations to facilitate the elaboration of the following sections. To avoid complex notations, given T with its corre-

sponding span partition $\vec{\vartheta}(T, \{t_1, t_2, \ldots, t_r\})$, we rewrite all the starting index of the subspans in $\vec{\vartheta}$ using a $2r+1$ dimensional vector $\vec{I}$, [5] where $I_0 = \vartheta_{0_b} = 0, I_1 = \vartheta_{1_b} = t_{1_b}, I_2 = \vartheta_{2_b} = t_{1_e} + 1, \ldots, I_{2r} = \vartheta_{2r_b} = t_{r_e} + 1$. In consequence, $\vec{\vartheta}$ can be rewritten with $\vec{\vartheta}(T, \{t_1, t_2, \ldots, t_r\}) = ([I_0, I_1), [I_1, I_2), [I_2, I_3), \ldots, [I_{2r}, l))^T$. Similarly, given T' with its corresponding span partition $\vec{\vartheta'}(T', \{t'_1, t'_2, \ldots, t'_r\})$, $\vec{\vartheta'}$ can be rewritten using a vector $\vec{J}$ as $\vec{\vartheta'}(T', \{t_1, t_2, \ldots, t_r\}) = ([J_0, J_1), [J_1, J_2), [J_2, J_3), \ldots, [J_{2r}, l'))^T$, where $J_0 = \vartheta'_{0_b} = 0, J_1 = \vartheta'_{1_b} = t'_{1_b}, J_2 = \vartheta'_{2_b} = t'_{1_e} + 1, \ldots, J_{2r} = \vartheta'_{2r_b} = t'_{r_e} + 1$.

Next we will present the recursive functions for aTSKs. Note that we only present the functions to evaluate the kernels within the same subspan, i.e. to evaluate $K_m(i, j)$ for all $(i, j) \in \{(i, j) | \forall p, 0 \leq p \leq 2r, i \in [I_p, I_{p+1}) \text{ and } j \in [J_p, J_{p+1})\}$. The kernel function covering the whole spans of T and T' can be evaluated by conducting the subspan kernel evaluation sequentially from the 0-th subspan to the $2r$-th subspan. Note that directly evaluating the kernel functions over the whole spans of T and T' will incur $l \times l'$ computational cost. Instead, by means of the characteristic of the subspan correspondence of aTSKs, we evaluate the kernel on each pair of *corresponded* subspans sequentially, which will help reduce the computational load especially when the number of anchored structures r is large.

Generally, we adapt the algorithms of TSKs to aTSKs by constraining the indices of the kernel functions to a subspan. However, there are two special cases we need to carefully deal with.

First, when the node pair n and n' are matched, in TSKs we need to roll back to $(n_b - 1, n'_b - 1)$ and consult the kernel value evaluated at $(n_b - 1, n'_b - 1)$. However, in aTSKs, this can be only rational when $n_b - 1$ and $n'_b - 1$ are in the *corresponded* subspans, i.e. $(n_b - 1) \in [I_p, I_{p+1})$ and $(n'_b - 1) \in [J_p, J_{p+1})$. When $n_b - 1$ and $n'_b - 1$ are *not* in the *corresponded* subspans, i.e. $(n_b - 1) \in [I_p, I_{p+1})$ and

[5] Note that $\vec{I}$ should be related to a given tree T with specified anchored subtrees $\{t_1, t_2, \ldots, t_r\}$, we do not explicitly integrate these parameters into $\vec{I}$ to simplify the presentation.

$(n'_b - 1) \in [J_q, J_{q+1})$ while $p \neq q$, it is infeasible to evaluate the kernel functions at $(n_b - 1, n'_b - 1)$.

Second, when the kernel functions are evaluated at the beginning of a subspan in either tree, for some functions, it may require to consult the previous index by decrement the current index. However, decrementing the index at the boundary leaves one of the consulted index in the previous span and the other index still in the current span. Again, we have to face the problem of evaluating the kernel values in *noncorresponded* subspans, which is disallowed in aTSKs.

To solve the above issue of the inconsistency of subspans, we need to appropriately design the kernels to allow the subspans to correctly communicate. This can be achieved by rolling back a little farther from the spot of the *noncorresponded* subspans to the *corresponded* subspans and use the previously evaluated kernel values in the *corresponded* subspans.

Before further elaboration, we define certain notations. Let $\psi_{i,j}$ refer to the set of *nonempty* subspan indices, where $\psi_{i,j} = \{p|\forall 0 \leq u < i, 0 \leq v < j, u \in [I_p, I_{p+1})$ and $v \in [J_p, J_{p+1})\}$. Briefly, $\psi_{i,j}$ is the set of indices for all the *corresponded nonempty* subspans bounded by indice i and j. In addition, let $\hat{p}(i,j)$ refer to the maximum value in the set $\psi_{i,j}$.

In fact, $\hat{p}(i,j)$ is the index of the *corresponded* subspans we are rolling back to, if i and j are in the *noncorresponded* subspans.

Based on the above discussion, we modify TSKs as follows:

4.5.2 Penalizing Length of spans (γ)

By adapting the algorithms from TSKγ, aTSK with penalty factor γ can be evaluated as follows. For all $(i,j) \in \{(i,j)|\forall p, 0 \leq p \leq 2r, i \in [I_p, I_{p+1})$ and $j \in [J_p, J_{p+1})\}$:

$$K_m^{(l)}(i,j) = 0 \qquad \text{if } \min(i,j) < m-1 \text{ for all } l \in \{1,2,3\}$$

$$K_m(i,j) = 0 \qquad \text{if } \min(i,j) < m-1$$

$$K_m^{(2)}(i,j) = 1 \qquad \text{if } m = 0$$

$$\Upsilon(i,j,n_b,n_b') = \begin{cases} \gamma^{l(n)+l(n')} K_{m-1}^{(2)}(n_b-1, n_b'-1) & \text{if} \quad \begin{array}{c}\exists q \in \psi(i,j),\\ n_b \in (I_q, I_{q+1}),\\ n_b' \in (J_q, J_{q+1})\end{array} \\ \gamma^{i-I_{\hat{p}(n_b,n_b')+1}+j-J_{\hat{p}(n_b,n_b')+1}+2} \cdot & \\ K_{m-1}^{(2)}(I_{\hat{p}(n_b,n_b')+1}-1, J_{\hat{p}(n_b,n_b')+1}-1) & \text{elsif} \quad \begin{array}{c}\exists q \in \psi(i,j),\\ n_b = I_q, n_b' \in (J_q, J_{q+1})\\ \text{or } n_b \in (I_q, I_{q+1}), n_b' = J_q\\ \text{or } \ n_b = I_q, n_b' = J_q\end{array} \end{cases} \tag{4.36}$$

$$K_m^{(3)}(i,j) = \begin{cases} \sum_{n_e=i} \sum_{\substack{n_e'=j \\ n \doteq n'}} \Upsilon(i,j,n_b,n_b') \cdot \Lambda(n,n') & \text{if } j = J_p \\ \gamma K_m^{(3)}(i,j-1) + \sum_{n_e=i} \sum_{\substack{n_e'=j \\ n \doteq n'}} \Upsilon(i,j,n_b,n_b') \cdot \Lambda(n,n') & \text{otherwise} \end{cases} \tag{4.37}$$

$$K_m^{(2)}(i,j) = \begin{cases} \begin{pmatrix} \gamma^{i-I_{\hat{p}(i,j)+j}-J_{\hat{p}(i,j)}+2} \cdot \\ K_m^{(2)}(I_{\hat{p}(i,j)+1}-1, J_{\hat{p}(i,j)+1}-1) \end{pmatrix} + K_m^{(3)}(i,j) & \text{if } i = I_p \\ \gamma K_m^{(2)}(i-1,j) + K_m^{(3)}(i,j) & \text{otherwise} \end{cases} \tag{4.38}$$

$$K_m^{(1)}(i,j) = \begin{cases} \sum_{n_e=i} \sum_{\substack{n_e'=j \\ n \doteq n'}} \Upsilon(i,j,n_b,n_b') \cdot \Lambda(n,n') & \text{if } j = J_p \\ K_m^{(1)}(i,j-1) + \sum_{n_e=i} \sum_{\substack{n_e'=j \\ n \doteq n'}} \Upsilon(i,j,n_b,n_b') \cdot \Lambda(n,n') & \text{otherwise} \end{cases} \tag{4.39}$$

$$K_m(i,j) = \begin{cases} K_m(I_{\hat{p}(i,j)+1}-1, J_{\hat{p}(i,j)+1}-1) + K_m^{(1)}(i,j) & \text{if } i = I_p \\ K_m(i-1,j) + K_m^{(1)}(i,j) & \text{otherwise} \end{cases} \tag{4.40}$$

First, we propose Eq. 4.36 to consult previous kernel values. The *if* condition is the case when $n_b - 1$ and $n_b' - 1$ are in the *corresponded* subspans. The *elsif* condition denotes the case when they belong to *noncorresponded* subspans. Note that in the condition $n_b = I_q, n_b' = J_q$ of the second case, $n_b - 1$ and $n_b' - 1$ may be

in the *corresponded* subspans, i.e. the subspans just ahead of the current subspans accommodating n_b and n'_b. However, when the previous subspan is empty[6], it has to roll back one more subspan. This can happen to either n_b or n'_b, or even both. The indices it finally rolls back to is $(I_{\hat{p}(n_b,n'_b)+1}-1, J_{\hat{p}(n_b,n'_b)+1}-1)$, where $\hat{p}(n_b,n'_b)$ refers to the index of the *corresponded* subspans it finally finds. In fact, $(I_{\hat{p}(n_b,n'_b)+1}-1, J_{\hat{p}(n_b,n'_b)+1}-1)$ refers to the pair of the ending index of this finalized *corresponded* subspans.

For other kernel functions, besides the component adapted from TSKγ, we need to consider the boundary case, which may incur a communication between subspans. In Eq. 4.37 and Eq. 4.39, we modify Eq. 4.24 and Eq. 4.26 in TSKγ by specializing the case of $j = J_p$. In Eq. 4.38 and Eq. 4.40, we modify the corresponding kernel functions by specializing the case of $i = I_p$.

4.5.3 Penalizing Count of Matched Subtrees (μ)

By adapting the algorithms from TSKμ, aTSK with penalty factor μ can be evaluated as follows. For all $(i,j) \in \{(i,j)|\forall p, 0 \le p \le 2r, i \in [I_p, I_{p+1}) \text{ and } j \in [J_p, J_{p+1})\}$:

$$
\begin{aligned}
&K_m^{(l)}(i,j) = 0 \qquad \text{if } \min(i,j) < m-1 \text{ for all } l \in \{1,2\} \\
&K_m^{(1)}(i,j) = 1 \qquad \text{if } m = 0 \\
&\Upsilon(i,j,n_b,n'_b) = \begin{cases}
K_{m-1}^{(1)}(n_b-1, n'_b-1) & \text{if} \quad \begin{array}{c} \exists q \in \psi(i,j), \\ n_b \in (I_q, I_{q+1}), \\ n'_b \in (J_q, J_{q+1}) \end{array} \\
K_{m-1}^{(1)}(I_{\hat{p}(n_b,n'_b)+1}-1, J_{\hat{p}(n_b,n'_b)+1}-1) & \text{elsif} \quad \begin{array}{c} \exists q \in \psi(i,j), \\ n_b = I_q, n'_b \in (J_q, J_{q+1}) \\ \text{or } n_b \in (I_q, I_{q+1}), n'_b = J_q \\ \text{or } n_b = I_q, n'_b = J_q \end{array}
\end{cases}
\end{aligned}
\tag{4.41}
$$

[6]Remember we allow the subspan between anchored structures to be possibly empty.

$$K_m^{(2)}(i,j) = \begin{cases} \sum_{n_e=i} \sum_{\substack{n'_e=j \\ n \doteq n'}} \Upsilon(i,j,n_b,n'_b) \cdot \Lambda(n,n') & \text{if } j = J_p \\ K_m^{(2)}(i,j-1) \ + \sum_{n_e=i} \sum_{\substack{n'_e=j \\ n \doteq n'}} \Upsilon(i,j,n_b,n'_b) \cdot \Lambda(n,n') & \text{otherwise} \end{cases} \tag{4.42}$$

$$K_m^{(1)}(i,j) = \begin{cases} K_m^{(1)}(I_{\hat{p}(i,j)+1}-1, J_{\hat{p}(i,j)+1}-1) + K_m^{(2)}(i,j) & \text{if } i = I_p \\ K_m^{(1)}(i-1,j) + K_m^{(2)}(i,j) & \text{otherwise} \end{cases} \tag{4.43}$$

Similar to aTSKγ, we introduce Eq. 4.41 to consult previous kernel values according to different conditions. The *if* condition is the case when $n_b - 1$ and $n'_b - 1$ are in the *corresponded* subspans. The *elsif* condition denotes the case when they belong to *noncorresponded* subspans.

For other kernel functions, we need to consider the boundary case, which may incur a communication between subspans. In Eq. 4.42, we modify Eq. 4.28 by specializing the case of $j = J_p$. In Eq. 4.43, we modify Eq. 4.29 by specializing the case of $i = I_p$.

Note that Eq. 4.30 is unnecessary to be computed for subspans, since it is not used by other kernel functions. This function can be evaluated after traversing all subspans as

$$K_m(l-1,l'-1) = \mu^m \cdot K_m^{(1)}(l-1,l'-1) \tag{4.44}$$

4.5.4 Penalizing Count of Gaps (τ)

By adapting the algorithms from TSKτ, aTSK with penalty factor τ can be evaluated as follows. For all $(i,j) \in \{(i,j) | \forall p, 0 \leq p \leq 2r, i \in [I_p, I_{p+1}) \text{ and } j \in [J_p, J_{p+1})\}$:

$$K_m^{(l)}(i,j) = 0 \qquad \text{if } \min(i,j) < m-1 \ \text{ for all } l \in \{1,2,3,4\}$$

$$K_m^{(l)}(i,j) = 0 \qquad \text{if } m = 0 \ \text{ for all } l \in \{1,2,3\}$$

$$K_m^{(4)}(i,j) = 1 \qquad \text{if } m = 0$$

$$K_m^{(1)}(i,j) = \begin{cases} K_m(I_{\hat{p}(i,j)+1} - 1, J_{\hat{p}(i,j)+1} - 1) & \text{if } i = I_p \text{or } j = J_p \\ K_m(i-1, j-1) & \text{otherwise} \end{cases} \tag{4.45}$$

$$K_m^{(2)}(i,j) = \begin{cases} 0 & \text{if } j = J_p \\ K_m^{(2)}(i, j-1) + K_m^{(4)}(i, j-1) & \text{otherwise} \end{cases} \tag{4.46}$$

$$K_m^{(3)}(i,j) = \begin{cases} 0 & \text{if } i = I_p \\ K_m^{(3)}(i-1, j) + K_m^{(4)}(i-1, j) & \text{otherwise} \end{cases} \tag{4.47}$$

$$\Upsilon(i,j,n_b,n_b') = \begin{cases} \tau^2 K_{m-1}^{(1)}(n_b - 1, n_b' - 1) \\ +\tau K_{m-1}^{(2)}(n_b - 1, n_b' - 1) \\ +\tau K_{m-1}^{(3)}(n_b - 1, n_b' - 1) \\ +K_{m-1}^{(4)}(n_b - 1, n_b' - 1) & \begin{array}{c} \exists q \in \psi(i,j), \\ \text{if } n_b \in (I_q, I_{q+1}), \\ n_b' \in (J_q, J_{q+1}) \end{array} \\ \tau^2 K_{m-1}^{(1)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau K_{m-1}^{(2)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau^2 K_{m-1}^{(3)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau K_{m-1}^{(4)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) & \begin{array}{c} \text{elsif } I_{\hat{p}(n_b,n_b')+1} < n_b, \\ J_{\hat{p}(n_b,n_b')+1} = n_b' \end{array} \\ \tau^2 K_{m-1}^{(1)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau^2 K_{m-1}^{(2)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau K_{m-1}^{(3)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau K_{m-1}^{(4)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) & \begin{array}{c} \text{elsif } I_{\hat{p}(n_b,n_b')+1} = n_b, \\ J_{\hat{p}(n_b,n_b')+1} = n_b' \end{array} \\ \tau^2 K_{m-1}^{(1)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau K_{m-1}^{(2)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau K_{m-1}^{(3)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +K_{m-1}^{(4)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) & \begin{array}{c} \text{elsif } I_{\hat{p}(n_b,n_b')+1} < n_b, \\ J_{\hat{p}(n_b,n_b')+1} < n_b' \end{array} \\ \tau^2 K_{m-1}^{(1)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau^2 K_{m-1}^{(2)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau^2 K_{m-1}^{(3)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) \\ +\tau^2 K_{m-1}^{(4)}(I_{\hat{p}(n_b,n_b')+1} - 1, J_{\hat{p}(n_b,n_b')+1} - 1) & \begin{array}{c} \text{elsif } I_{\hat{p}(n_b,n_b')+1} = n_b, \\ J_{\hat{p}(n_b,n_b')+1} < n_b' \end{array} \end{cases} \tag{4.48}$$

$$K_m^{(4)}(i,j) = \sum_{n_e=i} \sum_{\substack{n'_e=j \\ n \doteq n'}} \Upsilon(i,j,n_b,n'_b) \cdot \Lambda(n,n') \tag{4.49}$$

$$K_m(i,j) \quad = \sum_{l=1}^{4} K_m^{(l)}(i,j) \tag{4.50}$$

Similarly, we propose Eq. 4.48 to consult previous kernel values. However, things get a little complicated when we penalize the count of gaps. In this case, we need to make sure whether there is a gap between the current index and the consulted index in the subspan rolled back to. In the first condition of Eq. 4.48, we evaluate the normal case when n_b and n'_b are not at the boundary of the subspans. In the second condition, there is a gap between the index n_b and the index it rolls back to, i.e. $I_{\hat{p}(n_b,n'_b)+1} - 1$, but there is no gap between n'_b and $J_{\hat{p}(n_b,n'_b)+1} - 1$. In the third condition, there is no gap between n_b and $I_{\hat{p}(n_b,n'_b)+1} - 1$, and there is no gap between n'_b and $J_{\hat{p}(n_b,n'_b)+1} - 1$ either. In the fourth condition, there is a gap between n_b and $I_{\hat{p}(n_b,n'_b)+1} - 1$, and there is a gap between n'_b and $J_{\hat{p}(n_b,n'_b)+1} - 1$ as well. In the last condition, there is no gap between n_b and $I_{\hat{p}(n_b,n'_b)+1} - 1$, but there is a gap between n'_b and $J_{\hat{p}(n_b,n'_b)+1} - 1$. These conditions are specialized so that the penalizing factor τ can be appropriately incurred.

For other kernel functions, we need to consider the boundary case to allow the subspans correctly communicate. In Eq. 4.46, the condition of $j = J_p$ is specialized. In Eq. 4.47, the condition of $i = I_p$ is specialized. In Eq. 4.45, both the conditions of $i = I_p$ and $j = J_p$ are specialized.

4.5.5 Mapping Kernels with Anchored Structures

In Section 3.1.2, we introduced the mapping kernel paradigm, which provides a simple necessary and sufficient condition to verify the positive semidefinite property for kernels (Shin and Kuboyama, 2008). The mapping kernel paradigm constructs a mapping system $\mathbb{M}$ (Eq. 3.4), which constrains the feature space of the kernel

to be a subspace of some original feature space. The mapping kernel constructed on the mapping subspace is a valid kernel if and only if $\mathbb{M}$ is transitive (Eq. 3.5). In this section, we first show that aTSK is a valid positive semidefinite kernel by means of the mapping kernel paradigm. In addition, we also propose some other mapping systems based on aTSK.

We follow the similar technique as shown in Section 4.4.1 to construct aTSK through the mapping kernel paradigm. The kernel function of aTSK can be written in the form that

$$\widetilde{K}_m(T, T') = \sum_{(u,u') \in \mathcal{M}_{T,T'}} \widetilde{k}_m(u, u') \tag{4.51}$$

where $\widetilde{k}_m(u, u')$ is the underlying kernel stated in Eq. 4.21, whose input domain is the *non-overlapped* node sequences. In addition, $\mathcal{M}_{T,T'}$ refers to a finite and symmetric set satisfying that

$$\mathbb{M} = (\{\mathcal{T} | T, T' \in \mathcal{T}\}, \mathcal{U}_T^m, \{\mathcal{M}_{T,T'} \subseteq \mathcal{U}_T^m \times \mathcal{U}_{T'}^m | \forall (u, u') \in \mathcal{M}_{T,T'} \ , \ u \doteq u' \text{and } u \backsim u'\}) \tag{4.52}$$

where $u \backsim u'$ denotes that the corresponding nodes in u and u' are in the *corresponded* subspans according to the tree structure partition of T and T'. Since the partition is universally defined for all trees, it is easy to see that $u \backsim u'$ is transitive. Therefore, $\mathbb{M}_{\text{aTSK}}$ is transitive under the constraints of $u \doteq u'$ and $u \backsim u'$. As a result, aTSK is a valid positive semidefinite kernel.

Basically, aTSK can be considered as applying additional constraints on the original subtree sequence feature space with respective to the mapping kernel paradigm. The constraints often come from the kernel designer's prior knowledge on the related tasks. In order to accomplish a valid kernel, the proposed constraints should satisfy the definition of mapping kernel that the mapping system is transitive.

Now we attempt to extend aTSK and propose additional mapping systems.

The key characteristic of these mapping systems is that they are able to highlight meaningful structure features in the original tree sequence feature space. Therefore, these constraints may offer additional meaningful features for certain tasks.

First, we propose the mapping system $\mathbb{M}_{\text{aTSK}_1}$, which applies the constraint that all the anchored structures should be embedded in the subtree sequence features. It's straightforward that $\mathbb{M}_{\text{aTSK}_1}$ is transitive, since the strict matching of all the anchored subtrees corresponds across different tree pairs. Therefore, the kernel aTSK_1 under this mapping system is a valid positive semidefinite kernel.

Second, we propose the mapping system $\mathbb{M}_{\text{aTSK}_2}$, which applies the constraint that at least one of the anchored structures is embedded in the subtree sequence features. However, $\mathbb{M}_{\text{aTSK}_2}$ is not transitive. This can be verified by a negative example. Given three tree instances, i.e. T with anchored subtrees t_1, t_2, T' with anchored subtrees t'_1, t'_2, T'' with anchored subtrees t''_1, t''_2, it is assumed that t_1 and t'_1 are matched, while t_2 and t'_2 are not matched; t'_2 and t''_2 are matched, while t'_1 and t''_1 are not matched. In addition, for any subtree sequences v from T, v' from T' and v'' from T'', under $\mathbb{M}_{\text{aTSK}_2}$, suppose we have

$$\begin{aligned} (v, v') &\in \mathcal{M}_{T,T'} \\ (v', v'') &\in \mathcal{M}_{T',T''} \end{aligned}$$

However, $(v, v'') \notin \mathcal{M}_{T,T''}$, since neither of the two anchored subtrees for T and T'' are matched. As a result, aTSK_2 is not a valid positive semidefinite kernel. Although, in this case the data can be separated in the pseudo Euclidean space in this case, the solution may be only a local optimum (Haasdonk, 2005).

In literature, engineering novel parse tree kernels often corresponds to the necessity of certain tasks. In order to appropriately constrain the feature space to achieve the optimal performance for a given task, the original tree structures are often modified to highlight the most differential features. For example, in semantic relation extraction, Zhang, Zhou, and Aw (2008) tailored the original parse

tree structure into several variants and showed that the path enclosed tree(PT) achieves the best performance. In shallow semantic parsing, not only the tailoring approach is employed (Moschitti, 2004; Moschitti, Pighin, and Basili, 2006a; Moschitti, Pighin, and Basili, 2008), nodes in the parse tree can also be marked in order to differentiate the structures over arguments between positive and negative instances (Moschitti, Pighin, and Basili, 2006b; Moschitti, Pighin, and Basili, 2008).

Unlike the previous works to design new kernels by explicitly modifying the parse trees, the proposed TSKs introduce novel feature representations of subtree sequence and can be employed in any tailored tree structures other than the original parse trees. Perhaps the objective of aTSKs, i.e. to constrain the original feature space into a refined space by localizing the features, is much more close to that of previous works. Previous works achieve this objective by the node marking method, i.e. when applying to the objective of aTSK, nodes in different segmentations can be marked with different tags and the standard algorithms for tree (sequence) kernel are used for computation. However, we claim that aTSK is more efficient since it can match the nodes by segmentation, therefore, avoiding traversing the entire $N_T \times N_{T'}$ node space. This improvement is especially beneficial for localizing subtree sequence based features, since it may heavily reuse the kernel values evaluated in previous segmentations.

4.6 Kernels over Multiple Parse Trees

In this section, we propose kernels over multiple parse trees. Previously, kernels are designed over a single tree and structure features explored by the kernels are adopted to compute the similarity between a pair of single trees. In this section, we extend the kernels from one single tree to multiple trees. In other words, the input instance is no longer a single parse tree, but multiple parse trees. This could

be very helpful to NLP tasks over multilingual applications, of which the input domain consists of multilingual parallel parse trees with each tree corresponds to a different language.

We first illustrate the kernel for the most simple case, i.e. Collins and Duffy's tree kernel over two parse trees, namely Bilingual Tree Kernels (BTKs). After that, we will generalize the kernel to the normal case with multiple parse trees and subtree sequence features.

Before elaborating the concepts of BTKs, we first illustrate some notations to facilitate further understanding. Given a pair of trees S and T, we refer S as the source tree and T as the target tree. Each tree pair $(S \cdot T)$ can be explicitly decomposed into multiple substructures which belong to the given substructure spaces. $\mathcal{S} = \{s_1, \ldots, s_i, \ldots, s_{|\mathcal{S}|}\}$ refers to the source substructure space; while $\mathcal{T} = \{t_1, \ldots, t_j, \ldots, t_{|\mathcal{T}|}\}$ refers to the target substructure space. A substructure pair (s_i, t_j) refers to an element in the set of the cross product of the two substructure spaces: $(s_i, t_j) \in \mathcal{S} \times \mathcal{T}$. Other notations follow the same definitions in Section 3.1.3.

4.6.1 Independent Bilingual Tree Kernel (iBTK)

Given the substructure spaces $\mathcal{S}$ and $\mathcal{T}$, we construct two vectors using the integer counts of the source and target substructures:

$$\mathbf{\Phi}(S) = \big(\#(s_1), \ldots, \#(s_i), \ldots, \#(s_{|\mathcal{S}|})\big)^T \tag{4.53a}$$

$$\mathbf{\Phi}(T) = \big(\#(t_1), \ldots, \#(t_j), \ldots, \#(t_{|\mathcal{T}|})\big)^T \tag{4.53b}$$

$$\mathbf{\Phi}(S \cdot T) = \{\mathbf{\Phi}(S), \mathbf{\Phi}(T)\} = \big(\#(s_1), \ldots, \#(s_{|\mathcal{S}|}), \#(t_1), \ldots, \#(t_{|\mathcal{T}|})\big)^T \tag{4.53c}$$

where $\#(s_i)$ and $\#(t_j)$ are the numbers of occurrences of the substructures s_i and t_j. The feature space of iBTK is designed to be the concatenation of the source feature vector and target feature vector as shown in Eq. 4.53c. In order to compute the dot product of the feature vectors in the exponentially high dimensional feature

space, we introduce the tree kernel functions as follows:

$$K_{iBTK}(S \cdot T, S' \cdot T') = K_t(S, S') + K_t(T, T') \tag{4.54}$$

The iBTK is defined as a composite kernel consisting of a source tree kernel and a target tree kernel which measures the source and the target structural similarity respectively. K_{iBTK} is a valid kernel due to the property that kernels are closed under the operation of direct sum (Eq.2.27). Therefore, the composite kernel can be evaluated using Collins and Duffy's tree kernel (Eq.3.21) for $K_t(S, S')$ and $K_t(T, T')$ respectively.

4.6.2 Dependent Bilingual Tree Kernel (dBTK)

The iBTK explores the structural similarity of the source and the target trees respectively. The disadvantage of the iBTK may be the fact that it fails to capture the correspondence across the substructure pairs. As an alternative, we further define a kernel to capture the relationship across the source and target counterparts. As a result, we propose the dependent Bilingual Tree kernel (dBTK) to jointly evaluate the similarity across tree pairs by enlarging the feature space to the cross product of the two substructure sets.

The dBTK takes the source and the target substructure pair as a single feature and recursively calculate over the joint substructures of the given tree pair. We define the dBTK as follows:

Given the substructure space $\mathcal{S} \times \mathcal{T}$, we construct a vector using the integer counts of the substructure pairs to represent a tree pair.

$$\mathbf{\Phi}(S \cdot T) = \big(\#(s_1, t_1), \ldots, \#(s_1, t_{|\mathcal{T}|}), \#(s_2, t_1), \ldots, \#(s_{|\mathcal{S}|}, t_1), \ldots, \#(s_{|\mathcal{S}|}, t_{|\mathcal{T}|})\big)^T \tag{4.55}$$

where $\#(s_i, t_j)$ is the number of occurrences of the substructure pair (s_i, t_j).

It is infeasible to explicitly compute the kernel function by expressing the subtrees as feature vectors. In order to achieve convenient computation, we deduce the kernel function as follows.

$$
\begin{aligned}
K_{dBTK}(S \cdot T, S' \cdot T') &= \langle \mathbf{\Phi}(S \cdot T), \mathbf{\Phi}(S' \cdot T') \rangle \\
&= \sum_{k=1}^{|\mathcal{S} \times \mathcal{T}|} \left(\sum_{n_s \in N_S} \sum_{n_t \in N_T} I_k(n_s, n_t) \cdot \sum_{n'_s \in N_{S'}} \sum_{n'_t \in N_{T'}} I_k(n'_s, n'_t) \right) \\
&= \sum_{n_s \in N_S} \sum_{n_t \in N_T} \sum_{n'_s \in N_{S'}} \sum_{n'_t \in N_{T'}} \Lambda\left((n_s, n_t), (n'_s, n'_t)\right) \qquad (4.56) \\
&= \sum_{n_s \in N_S} \sum_{n_t \in N_T} \sum_{n'_s \in N_{S'}} \sum_{n'_t \in N_{T'}} \left(\Lambda(n_s, n'_s) \cdot \Lambda(n_t, n'_t)\right) \qquad (4.57) \\
&= \sum_{n_s \in N_S} \sum_{n'_s \in N_{S'}} \Lambda(n_s, n'_s) \sum_{n_t \in N_T} \sum_{n'_t \in N_{T'}} \Lambda(n_t, n'_t) \\
&= K_t(S, S') \cdot K_t(T, T') \qquad (4.58)
\end{aligned}
$$

The notations follow what have been defined in Section 3.1.3. The deduction from expression 4.56 to expression 4.57 is derived according to the fact that the number of identical substructure pairs rooted in the node pairs (n_s, n_t) and (n'_s, n'_t) equals to the product of the respective counts. As a result, the dBTK can be evaluated as a product of two single tree kernels. Here we verify the validity of the kernel by directly constructing the feature space for the inner product. A similar proof is given in (Moschitti and Zanzotto, 2008). Alternatively, this could be proved by the positive semidefinite characteristic of the tensor product of two kernels (Eq.2.28). The decomposition benefits the efficient computation to use the algorithm of Collins and Duffy's tree kernel (Eq.3.21).

The computational complexity of the both iBTK and dBTK is $O(|N_S| \cdot |N_{S'}| + |N_T| \cdot |N_{T'}|)$.

4.6.3 Generalized Kernels over Multiple Trees

Previously, we extend Collins and Duffy's tree kernel from one single tree to the case of two trees. Basically, the extensions come from the construction of the feature space by means of the source feature space and the target feature space independently or jointly. When constructing the feature space independently, i.e. iBTK, the resulting kernel is the direct sum of the source tree kernel and the target tree kernel. When constructing the feature space jointly, i.e. dBTK, the resulting kernel is the tensor product of the two single tree kernels. Based on these results, it is obvious to further extend the kernel to multiple trees as follows:

Given an input instance with k trees $\{T_1, \ldots, T_k\}$, the independent tree kernel (iTK) over the k-tree space can be evaluated as follows:

$$K_{iTK}(\{T_1, \ldots, T_k\}, \{T'_1, \ldots, T'_k\}) = K_t(T_1, T'_1) + \cdots + K_t(T_k, T'_k) \tag{4.59}$$

On the other hand, the dependent tree kernel (dTK) is evaluated by

$$K_{dTK}(\{T_1, \ldots, T_k\}, \{T'_1, \ldots, T'_k\}) = K_t(T_1, T'_1) \cdots K_t(T_k, T'_k) \tag{4.60}$$

Likewise, when extending the kernels over multiple trees by utilizing the subtree sequence features. The independent tree sequence kernel (iTSK) over the k-tree space can be evaluated as follows:

$$K_{iTSK}(\{T_1, \ldots, T_k\}, \{T'_1, \ldots, T'_k\}) = K_{ts}(T_1, T'_1) + \cdots + K_{ts}(T_k, T'_k) \tag{4.61}$$

Similarly, the dependent tree sequence kernel (dTSK) is evaluated by

$$K_{dTSK}(\{T_1, \ldots, T_k\}, \{T'_1, \ldots, T'_k\}) = K_{ts}(T_1, T'_1) \cdots K_{ts}(T_k, T'_k) \tag{4.62}$$

where $K_{ts}(T_i, T'_i)$ is the kernel function for the tree sequence based kernels evaluated on the i-th feature space.

Tree kernels over multiple parse trees have been attempt in certain tasks such as textual entailment (Zanzotto and Moschitti, 2006; Moschitti and Zanzotto, 2007;

Zanzotto, Pennacchiotti, and Moschitti, 2009), which employs a variant of dependent kernels and question/answer classification (Moschitti, 2008), which employ a variant of independent kernels.

4.7 Summary

In this section, we propose a series of kernels constructed on the different tree sequence feature spaces. Specially, cTSK evaluates the structure space of contiguous subtree sequences. TSKs allow additional features of non-contiguous subtree sequences. In addition, we verify the positive semidefinite property of the generalized TSK by showing that it can be constructed based on the mapping kernel paradigm. SSKs are proposed to facilitate the evaluation of TSKs, which help capture the root node sequence of the subtree sequence. Based on the mapping kernel paradigm, we further constrain the feature space of TSKs to more accommodate the anchored structures, which achieves aTSKs. aTSKs are expected to well facilitate the applications built on target relational constituents. By more aggressively constraining the feature spaces, we propose aTSK_1 and aTSK_2 and verify that aTSK_1 is a valid kernel while aTSK_2 is not. Three weighting schemes are proposed for TSK, SSK and aTSK. Based on these weighting schemes, we propose efficient algorithms to evaluate the corresponding kernels. The proposed kernels are then shown able to be extended to multiple parse trees.

Chapter 5

Tree Sequence Kernels for Single Parse Tree

In the following two chapters, the proposed series of kernels over parse trees will be applied in NLP applications. This chapter will target applications in monolingual tasks, while the next chapter will focus on multilingual tasks, i.e. tasks making use of multiple parse trees from different languages. Generally, tree kernels have been successfully applied in many monolingual NLP tasks such as syntactic parsing (Collins and Duffy, 2002), question classification (Zhang and Lee, 2003; Moschitti, 2006), relation extraction (Zhang, Zhou, and Aw, 2008), pronoun resolution (Yang, Su, and Tan, 2006), textual entailment (Zanzotto and Moschitti, 2006) and semantic parsing (Zhang et al., 2008a; Moschitti, 2004). Among them, two of the well studied tasks are Question Classification and Relation Extraction, for which various extensions of tree kernels on different tree structures, i.e. phrase based and dependency based, have been developed. As a result, we select these two tasks to verify the effectiveness of the tree sequence based kernels proposed in the last Chapter.

This chapter is organized as follows: The related work for both tasks will

be discussed in the following section. In Section 5.2, the detailed experimental results are presented, where the proposed tree sequence based kernels are compared with Duffy and Collin's tree kernel as well as the partial tree kernel (Moschitti, 2006). This chapter is concluded after a discussion of the experimental results across different applications.

5.1 Background and Related Work

Question Classification is a crucial subtask for implementing question answering systems. Basically, given a question sentence, it is necessary to identify what specific issues the question concerns. For example, given a question "*Who is Google's founding CEO?*", an ideal question classifier should identify that this question is asking for a person's name rather than a date or a location. An accurate and complete analysis of the question may effectively help a question answering system achieve a good answer extraction and selection, as well as generating more precise answers.

By considering question classification as a multi-classification task, many machine learning techniques have been applied in this problem. Radev et al. (2005) applied decision rule learning with set-valued features in this task. The features employed in this approach are augmented to have a set of values instead of a single value as in the traditional decision tree. Li and Roth (2002) used the Sparse Network of Winnows (SNoW) to classify questions. They adopted words, part-of-speech tags, chunks, name entities, head chunks and related words as their features. Their taxonomy of questions is under 6 coarse types and 50 fine classes. They also released their $5,500$ training questions and 500 testing questions, which have benefited a lot of later research on this topic.

Kernel based methods have also been applied in this task and been verified to outperform the above feature based methods. Zhang and Lee (2003) employed

Collins and Duffy's tree kernel with SVM as the kernel machine for question classification. The slight difference in between is that their kernel allows to match the single terminal (leaf) node as valid tree structures. Experimental results suggest that the tree kernel based method with SVM not only outperforms other machine learning techniques such as Nearest Neighbors, Naive Bayes, Decision Trees and SnoW, but also outperforms the linear kernels constructed by bag-of-words and bag-of-ngrams with SVM. Suzuki et al. (2003) adopted the hierarchical directed acyclic graph kernel (Suzuki, Sasaki, and Maeda, 2006) with SVM as the kernel machine for this task. They used a much larger taxonomy consisting of 150 question classes for the Japanese questions. They designed four feature sets by means of words, name entities and lexical information. The graph kernel on each feature set outperforms the bag-of-words kernel and SNoW for a large margin, which suggests the structure information plays an essential role for this task. Bloehdorn and Moschitti (2007) modified Collins and Duffy's tree kernel by incorporating semantic knowledge of term similarities. This kernel allows flexible matching between tree fragments with lexical nodes. Experimental results suggest that by combining both syntactic and semantic information, this kernel achieves better performance than traditional syntactic tree kernels.

Relation Extraction is a task to find various semantic relations between entity pairs in the document. For example, the sentence "*Larry Page was Google's founding CEO*" conveys a semantic relation of "EMPLOYMENT.executive" between the entity pairs "*Larry Page*" (entity type: person) and "*Google*" (entity type: company). Relation extraction is a challenging task in information extraction and essential for deep information understanding and management. Similar to question classification, both feature based methods and kernel based methods have been applied in this task.

For the feature based methods, Miller et al. (2000) incorporated the semantic

knowledge corresponding to entity pairs and relations into syntactic parse trees by the tree augmentation algorithm. Then a statistical generative model similar to the generative parsing algorithms (Collins, 1997) was employed to perform both entity and relation recognition. Kambhatla (2004) proposed various syntactic features and integrate those features into a Maximum Entropy (ME) model. Based on this work, Zhou and Zhang (2007) further introduced syntactic features from phrase chunking and semantic features using WordNet and name list. Instead of the ME model, they adopted SVM as the classifier. Experimental results suggest that these additional features are very effective. Zhao and Grishman (2005) also employed diverse linguistic features, both syntactic and semantic. However, instead of using the features as a single feature vector, they formalized each class of features as a single kernel and combined these kernels as a composite kernel. The composite kernel allows more flexibility to weight different types of features.

Tree kernel based methods have also been well studied for relation extraction. Zelenko, Aone, and Richardella (2003) integrated sequence kernel in a minimum shallow parse tree that covers the two entities. The kernel matches the subsequence child nodes of a given node. Their experiments focus on detecting person-affiliation and organization-location relations. This kernel was further extended and applied on dependency trees by Culotta and Sorensen (2004). In this work, more syntactic and semantic knowledge is introduced, i.e. part-of-speech tags, chunking tags, entity types, entity levels, WordNet hypernyms and relation-argument tags. This information is formed as a feature vector for each node in the dependency tree. Since the parse trees are matched recursively from root down to the leaf nodes, it requires the tree structure of the relation instances to be similar. Otherwise the recall may be low. Bunescu and Mooney (2005) propose a shortest path kernel on dependency structures based on the argument that the key information deciding the relations is mainly collected by the path between the entities in the dependency structure. The

kernel is evaluated by counting the number of common word classes at each node along the paths. However, they limited the two paths to be strictly the same length. Therefore, although it achieves better performance than the kernel proposed by Culotta and Sorensen (2004), it still suffers from the low recall. Reichartz, Korte, and Paass (2009) addressed the restriction of the above two dependency kernels by allowing the matching of every combination of nodes in the given tree pair as well as allowing all possible subsequences of nodes along the shortest paths to be matched. Experimental results on ACE corpora (Doddington et al., 2004) suggest the relaxation of the constraints benefit both precision and recall. However, the above reported kernels on dependency trees showed lower performance than the feature based methods (Zhou and Zhang, 2007) on ACE corpora.

In order to better utilize the structure features embedded in the parse trees, Zhang, Zhou, and Aw (2008) applied Collins and Duffy's tree kernel on the phrase parse trees. Their experiments reveal that by appropriately designing the feature space explored by the tree kernels, the performance significantly improves against the dependency tree kernels and the feature based methods on ACE 2003 and 2004 corpora. They also showed that the Shortest Path-enclosed Tree (PT) performed best among the different tree structures. Nevertheless, one of the problems for PT is that it is unable to capture the context information outside the shortest paths, which may be useful for relation extraction. To verify this point, Zhou et al. (2007) gathered some statistics on a small sample of ACE 2003 corpus. By categorizing the relation instances into five types, they showed that the context information is quite important for certain types of instances. In order to better use the context information outside the PT structure, they also proposed a context sensitive tree kernel to match the context around a subtree. The ancestor node path of a subtree is considered as their context.

Based on previous tree based kernels, Nguyen, Moschitti, and Riccardi (2009)

proposed various composite kernels to combine the advantages of the different convolution kernels on different tree structures (phrase tree and dependency tree), along with sequence kernels to capture semantic information. The composite kernels can well excel the advantages of individual kernels and achieve the state-of-the-art performance for relation extraction on ACE corpora.

The above discussion suggests that for both tasks the parse tree kernels can outperform the feature based methods as long as the substructure spaces explored by the kernels are appropriately designed. On the other hand, the parse tree may not have been fully explored in the previous work, regardless of the approaches they adopted, either feature based or kernel based. Although this deficiency has been implicitly overcome by relaxing certain strict matching constraints or conducting composite kernels from previous well designed kernels, it is still more attractive to explore more meaningful substructures to benefit NLP applications. It is expected that the tree sequence structures, both contiguous and non-contiguous, may provide more information for recognizing the semantics behind the plain text. As a result, the effectiveness of tree sequence based kernels will be verified on both tasks in the following sections.

5.2 Experiments

To assess the effectiveness of the tree sequence based kernels proposed in Chapter 4, we apply the TSKs in two NLP applications, i.e. Question Classification and Relation Extraction.

Both tasks are addressed as a multi-class classification problem. For the multi-class applications, we use the one vs. others strategy to select the optimal class with the largest margin. To be consistent with previous works, the experimental results of Question Classification is reported with Accuracy, while for Relation Extraction, evaluations are conducted based on three metrics, i.e. Precision (**P**), Recall (**R**)

and F-measure (**F**). In our implementation, tree sequence kernels are integrated into the tree kernel tool[1] (Moschitti, 2006) based on the SVM classifier (Joachims, 1999).

For both tasks, we present the results of the tree kernel (Collins and Duffy, 2002) and Partial Tree Kernel (PTK) (Moschitti, 2006) on the phrase parse trees as the two baselines. Along with the TSKs, all the kernels are normalized to remove the bias caused by different sizes as follows:

$$\widehat{K}_{t(s)}(T, T') = \frac{K_{t(s)}(T, T')}{\sqrt{K_{t(s)}(T, T) \cdot K_{t(s)}(T', T')}} \tag{5.1}$$

5.2.1 Question Classification

In the experiment of question classification, we follow the standard experimental setup in the previous work (Moschitti, 2006) as follows: (1) we conduct the classification task on coarse grained question taxonomy with six major categories: Abbreviations, Descriptions, Entity, Human, Location and Number; (2) we also use the same data (Li and Roth, 2002). The corpus consists of 5500 labeled instances for training and 500 instances for testing. (3) Stanford parser is adopted to generate the phrase parse tree structures (Klein and Manning, 2003).

We conduct experiments by using TSKμ, TSKτ and cTSK to compare the effectiveness of different structure representations. The experimental results are also compared with the two baselines of Collins and Duffy's tree kernel and PTK. Additionally, since cTSK can be considered as syntactic structures over lexical sequences with arbitrary length, we also compare the proposed tree sequence based kernels with the polynomial kernels ($d = 2$) on Bag-of-Ngrams (BoN) and Bag-of-words respectively. This may reveal the effectiveness of TSKs in capturing the structure features over lexical sequences.

[1] http://disi.unitn.it/moschitti/Tree-Kernel.htm

	1k	2k	3k	4k	5.5k
BoW	78.0	84.0	85.8	86.6	87.2
BoN	74.8	81.6	82.8	86.4	88.0
TK(Collins and Duffy, 2002)	83.8	87.8	90.2	89.4	91.0
PTK(Moschitti, 2006)	81.4	86.8	89.4	89.4	90.8
cTSK	84.6	88.4	89.6	90.8	**92.2**
TSKμ	83.6	87.4	90.0	89.6	**91.4**
TSKτ	84.6	88.4	90.0	91.2	**92.4**

Table 5.1: Accuracy of Question Classification

All the experiments are conducted on various sizes of training data from $1k$ to the whole set with the same division in the original data set. We empirically set $C = 10$ for the SVM classifier. The penalizing factors for TK, PTK, TSKμ, TSKτ and cTSK are optimized by the 5-fold cross validation on the $5.5k$ training corpus. Finally, we choose the parameters $\lambda = 0.2$ for TK; $\hat{\mu} = 0.7$[2] and $\lambda = 0.1$ for PTK; $\mu = 0.6$ and $\lambda = 0.2$ for cTSK; $\lambda = 0.25$ and $\mu = 0.05$ for TSKμ; $\lambda = 0.15$ and $\tau = 0.25$ for TSKτ.

The experimental results are presented in Table 5.1. We analyze the results from two perspectives:

Using the whole training data:

- It can be observed that cTSK and the TSKs outperform TK and PTK when using the entire training corpus. Specially, TSKτ achieves the optimal performance. It outperforms TK by 1.4 point of accuracy and outperforms PTK by 1.6 point. This indicates the advantage of the additional tree sequence features in question classification. Additionally, cTSK outperforms TK and PTK by 1.2 point and 1.4 point of accuracy respectively. This verifies the additional benefit brought by

[2]Besides the factor λ to penalize the number of production rules, PTK also employs the decay factor $\hat{\mu}$ to penalize the length of the child sequence. Readers may refer to Moschitti (2006) for more details of this parameter.

the tree sequence structure features mostly comes from the contiguous tree sequences and the improvement from the non-contiguous tree sequences could be very limited. However, TSKμ outperforms TK and PTK by only a little amount of accuracy when using larger training data ($4k$ & $5.5k$), which suggest that penalizing the number of gaps for the non-contiguous tree sequences is better than penalizing the number of subtrees for question classification.

- In addition, PTK achieves slightly lower performance than TK. This finding is consistent with the previous results in Moschitti (2006), which suggests that PTK is slightly less accurate than TK on phrase parse trees.
- Furthermore, both TK and the cTSK/TSKs achieve better performance than the polynomial kernel only using lexical information (BoN/BoW). This indicates the syntactic features in phrase parse tree are very useful for detecting question types.

Using small training data:

- By using only a part of the training data, we find that most of the results are consistent with the above findings by using the entire training data, except when the data volume increases from $2k$ to $3k$. When using 3k data, cTSK outperforms PTK but underperforms TK. At the same time, we find that when using 3k data, the result of BoN is also unusual. The improvement of $3k$ against $2k$ is rather lower (+1.2) than using other amount ($2k/1k + 6.8; 4k/3k + 3.6; 5k/4k + 3.6$). It seems the contiguous lexical features in the additional 1k of the 3k data are not very effective. This may also account for the lower performance of cTSK in 3k data.
- In addition, when using small training data ($1k$ & $2k$), TSKμ underper-

forms TK. It's easy to attribute this result to the fact that the additional large structures matched by TSKs tend to be very sparse in small data. However, we also notice that TSKτ outperforms TK in small data. The inconsistency between the performance of TSKμ and TSKτ in small data when compared with TK indicates that the tree sequence with many gaps may not well contribute to the classification accuracy and the penalty of τ is essential for penalizing those structures.

5.2.2 Relation Extraction

The experimental settings of relation extraction follow Zhang, Zhou, and Aw (2008), since it reports the stat-of-the-art performance using tree kernels. (1) We use the same corpus (ACE 2004, LDC2005T09) with 348 documents and 4400 relation instances. The corpus consists of 7 types of entities, 7 major relation types and 23 relation subtypes. (2) We also employ Charniak parser (Charniak, 2001) to generate the parse tree. (3) We extract all entity mentions appearing in the same sentences as relation candidate pairs.

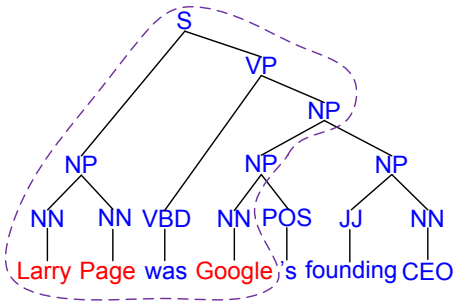
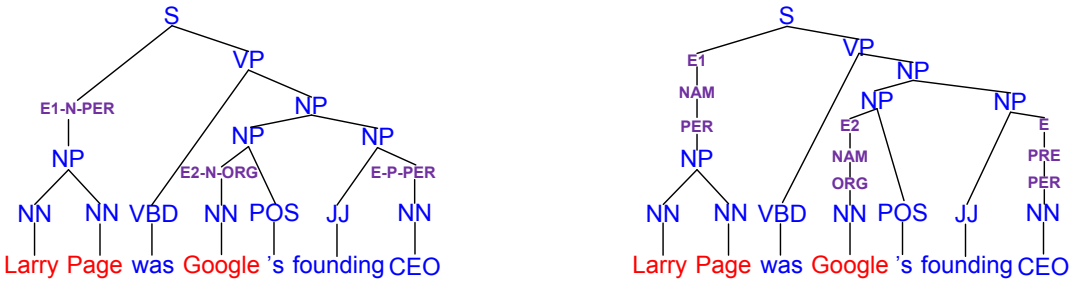


Figure 5.1: Parse tree instance for relation extraction

Zhang, Zhou, and Aw (2008) reported that using Collins and Duffy's tree kernel on the shortest path-enclosed tree (PT, the dashed line area in Fig. 5.1, given "*Larry Page*" and "*Google*" are identified as entities.) outperforms using it

on the complete constituent tree (MCT[3], the entire subtree shown in Fig. 5.1) by a large margin. They attributed this observation to the possibility that the left and right context information of MCT may bring noisy features. However, Zhou et al. (2007) verified that the context information benefited certain relation types by analyzing a sample in ACE 2003 corpus. Therefore, it is quite possible that the poor performance of MCT stems from the lack of capability of tree kernel to capture the context information provided by MCT rather than the ineffectiveness of the context. On the other hand, TSKs are designed to capture more meaningful structure information, especially the structure features over large context.



(a) Flat integration

(b) Multiple layer integration

Figure 5.2: Illustration of Tree Sequence Structures.

Originally, the candidate entities are encapsulated with five entity types (i.e. person, organization, location, facility and geo-political entity) and three entity mentions (i.e. names, nomial expressions and pronouns). We propose two methods to integrate the entity information by augmenting the parse trees with the entity annotations. One is to use the combination of the two levels of annotation as a flat grammar tag (*flat*). The other is to interpret it as a multi-layer tree structure (*multi*). As shown in Fig. 5.2(a), the flat representation of "*E1-N-PER*" denotes that the current phrase is the *1st* entity, its entity type is "*PERSON*" and its

[3]MCT is the minimal constituent rooted by the nearest common ancestor of the two entities under consideration while PT is the minimal portion of the parse tree (may not be a complete subtree) containing the two entities and their internal lexical words.

mention level is "*NAME*", and likewise for the second entity. On the contrary, the multi-layer representation integrate the same information in the same parse tree as multiple levels as shown in Fig. 5.2(b).

In addition, the anchored tree sequence kernels (aTSKs) are designed to capture the context information around the anchored structures, which is expected to better facilitate relation extraction on the utilization of context features. Therefore, the experiments in this section will not only include the results on the PT structures, but also include the results on the MCT structures. The results are then compared between the proposed kernels and the baseline kernels on MCT and PT respectively.

For aTSKs, we only verify their effectiveness in the *multi*-layer style for integrating the entity information. Thus, the anchored structures are defined as the three-layer entity annotations excluding the structures below the entity types. For instance, the anchored structure of the previous example is the three-level subtree "(Entity (NAME ORG))" in the original tree structure. For the additional mapping system $\mathbb{M}_{\text{aTSK}_1}$ and $\mathbb{M}_{\text{aTSK}_2}$, the strict matching is conducted on the first two levels. In this case, the matching of "(Entity NAME)" is compulsory for aTSK_1 and aTSK_2.

In the experiment, we empirically set $C = 7$ for SVM. A small amount of data is used as the development set to tune the kernel parameters with respect to the F-score. In addition, all the experiments are done with a 10-folds cross validation on the remaining corpus. In each experiment, the corpus is divided into 10 equal sized portions. In each fold, one of the portions is selected to be the testing data and the remaining to be the training data. In case of the statistical significance test is required, a paired Student's t-test is carried out at 0.05 level of significance.

The results on MCT and PT are presented in Table 5.2 and Table 5.3 respectively. We analyze the results from the following aspects:

- In both tables, we find that all TSKs (TSKγ,TSKμ,TSKτ) significantly out-

		P	R	F
flat	TK	66.64	58.76	62.42
	PTK	64.63	63.32	63.94
	cTSK	64.20	64.55	64.35
	TSKγ	66.44	63.87	65.11
	TSKμ	66.68	63.50	65.03
	TSKτ	66.57	64.13	65.31
mul	TK	65.82	62.85	**64.27**
	PTK	49.15	52.28	50.65
	cTSK	65.85	66.27	**66.03**
	TSKγ	66.75	66.64	66.67
	TSKμ	67.44	67.06	**67.22**
	TSKτ	66.77	66.71	66.71
mul	aTK	67.70	68.50	**68.08**
	aTSKγ	70.43	68.50	69.43
	aTSKμ	73.50	67.50	70.35
	aTSKτ	73.91	67.71	**70.65**
	aTSK$_1\gamma$	68.14	67.50	67.80
	aTSK$_1\mu$	68.37	64.22	66.21
	aTSK$_1\tau$	69.31	63.90	66.48
	aTSK$_2\gamma$	70.60	71.31	70.94
	aTSK$_2\mu$	68.70	69.36	69.01
	aTSK$_2\tau$	71.28	69.22	70.22

Table 5.2: Performance for Relation Extraction on MCT

		P	R	F
flat	TK	69.54	66.34	67.88
	PTK	67.93	65.66	66.76
	cTSK	68.89	68.36	68.60
	TSKγ TSKμ TSKτ	71.85	67.59	69.63[4]
mul	TK	67.11	69.70	**68.36**
	PTK	45.81	59.69	51.73
	cTSK	70.99	68.85	**69.88**
	TSKγ	71.95	68.45	70.13
	TSKμ	73.56	68.19	**70.76**
	TSKτ	73.11	68.33	70.62
mul	aTK	67.59	71.19	**69.32**
	aTSKγ	73.34	69.89	71.55
	aTSKμ	72.61	70.87	71.70
	aTSKτ	72.55	71.56	**72.03**
	aTSK$_1\gamma$	70.99	65.27	67.99
	aTSK$_1\mu$	70.61	65.48	67.93
	aTSK$_1\tau$	70.97	65.27	67.98
	aTSK$_2\gamma$	69.15	72.10	70.57
	aTSK$_2\mu$	69.62	71.38	70.47
	aTSK$_2\tau$	69.24	71.66	70.41

Table 5.3: Performance for Relation Extraction on PT

perform TK and PTK in F-score for all configurations (flat/multi and MCT/PT). Specifically, in the *multi*-layer style, TSK achieves the best F-score when penalizing the number of subtrees with factor μ for both MCT (67.22) and PT (70.76) structures. TSKμ achieves the improvement in F-score of +2.95 for MCT and +2.40 on PT against TK. In addition, cTSK in the *multi*-layer style significantly outperforms TK and PTK in F-score for both MCT and PT structures as well. Specifically, cTSK in the *multi*-layer style achieves the improvement in F-score of +1.76 for MCT and +1.52 for PT against TK. However, the improvement of TSKs against cTSK is not statistically signifi-

cant in all configurations.

- In addition, we also observe that the *multi*-layer representation is more effective to integrate the entity information for TSKs/cTSK/TK than the flat representation. Furthermore, compared with the flat integration, TSKs/cTSK gain more improvement against TK when using the *multi*-layer representation since it can introduce more structure information to benefit TSKs/cTSK.

- Due to the effectiveness of the *multi*-layer representation demonstrated by TSK, we further conduct experiments for the anchored tree sequence kernels (aTSKs) using the *multi*-layer style. From both tables, it can be observed that all aTSKs (aTSKγ,aTSKμ,aTSKτ) significantly outperform the corresponding TSKs in F-score for both MCT and PT structures. Therefore, to capture the tree sequence features with their relative positions against the entity structures can further benefit the identification of the entity relations. This observation suggests that the spatial relationship against the anchored entity pairs is very important to differentiate the structure features. However, the additional mapping systems of $\mathbb{M}_{\text{aTSK}_1}$ and $\mathbb{M}_{\text{aTSK}_2}$ do not show any advantages against the mapping system $\mathbb{M}_{\text{aTSK}}$. This result reveals that the additional constraints of matching the entity structures are so strict that the corresponding kernels may lose certain meaningful features across instances with different entity types.

- In addition, we also apply the similar mapping system of $\mathbb{M}_{\text{aTSK}}$ on the Collins and Duffy's tree kernel, which requires the single subtree structures to be

[4]By tuning the kernel parameters in the development set, we obtained the best values of γ,μ and τ for the respective kernels with *flat*-layer style on the PT structure all equal to 1. Therefore, no additional penalty is incurred for all the three kernels except the original λ factor. As a result, TSKγ,TSKμ,TSKτ achieves the same scores for this setting.

matched within the corresponding segment between the parse tree pair.[5] This kernel is denoted as "aTK" in both tables. For both MCT and PT, we can observe that aTK outperforms TK in the *multi*-layer representation, which further verifies the effectiveness of the constraint to match the structures according to their positions. At the same time, we can also detect the improvement of aTSKs against aTK for both MCT and PT. This also reinforces the advantage of the tree sequence structures over the single tree structures in relation extraction.

- Although cTSK and TSKs are able to capture more meaningful structure features by using the *multi*-layer integration of entity structures, it is not the case for PTK. When using the *multi*-layer representation, PTK suffers from a huge drop in performance. We may address this issue by comparing with TK, since both PTK and TK explore the single tree structures. Generally, PTK explores a much larger feature space. Therefore, the number of noisy features in PTK is much more than TK. By matching the entity information using multiple layers instead of a single tag, the number of noisy features is further enlarged, i.e. when the noisy structure and the *multi*-layer entity structure are concatenated to form a single tree. Therefore, the noisy features may gain more weight to play an essential role, which will limit the impact of the good features. In order to verify the above observation and explanation, we further constrain the feature space explored by PTK. Three variants of PTK (PTKc1, PTKc2, PTKc3) with constraint feature spaces are proposed.

[5]Note that the additional mapping systems of $\mathbb{M}_{\text{aTSK}_1}$ and $\mathbb{M}_{\text{aTSK}_2}$ are inappropriate to be applied in the single tree based kernels for relation extraction. That is because these two extensions require the anchored entities to be matched and the entities are possible to be non-contiguous. Therefore, the feature space of covering the two entities at the same time could be very sparse, which fails to achieve the goal of these extended kernels. As a result we do not apply the extended mapping systems on Collins and Duffy's tree kernel.

	P	R	F
PTK	49.15	52.28	50.65
PTKc1	60.66	55.12	57.73
PTKc2	61.89	57.95	59.84
PTKc3	63.72	57.58	60.47
TK	65.82	62.85	64.27

Table 5.4: Constraint PTKs on MCT

	P	R	F
PTK	45.81	59.69	51.73
PTKc1	61.09	57.78	59.35
PTKc2	68.40	57.78	62.37
PTKc3	55.83	61.48	58.49
TK	67.11	69.70	68.36

Table 5.5: Constraint PTKs on PT

PTKc1 constrains the feature space by allowing the child sequences to be match *iff* their parents have same number of child nodes. Based on PTKc1, PTKc2 further requires the matched child nodes have the same ordinal rank in the child sequences of their parents. PTKc3 matches the child sequences *iff* the production rules rooted at their parents are identical. As a result, the feature spaces of the variants are more constrained compared with original PTK while less constraint compared with TK. In other words, the structure spaces explored are the subsets of the original PTK, while are the supersets of TK. As shown in Table 5.4 and Table 5.5, the variants of PTK achieves better performance on both MCT and PT compared with PTK but less than TK, which suggests that the multiple level representation exemplifies the influence of the large amount of noisy features in PTK.

- Furthermore, by comparing Table 5.2 and Table 5.3, we find MCT suffers from a large performance drop against PT, which is also found in the experiments of Zhang, Zhou, and Aw (2008). Although this deficiency of MCT is still found in TSKs and aTSKs, the actual gap between MCT and PT is narrowing. This observation suggests that the tree sequence features are more differentiable so that the noisy features in the context of MCT are overwhelmed by those useful features.

- The best F-score (72.03) is achieved by using aTSKτ in the *multi*-layer style on PT. aTSKτ significantly improves the performance against the two baselines

TK (+3.67) with *multi* style and PTK (+5.27) with *flat* style, which reveals the advantages of the kernels which can highlight the anchored structures in tree sequences.

5.3 Discussion

Compared with the task of question classification, relation extraction gains more improvement by using TSKs/aTSKs. The effectiveness of TSKs/aTSKs on Relation extraction can be attributed to the fact that TSKs/aTSKs can capture multiple items in a non-contiguous tree sequence simultaneously. Relation Extraction is a task focusing on paired relational constituents, which are highly likely to be disjointed. Therefore, it will be useful to capture patterns consisting of information over both entities as a single structure, which cannot be captured by single tree based kernels. As shown in Fig 5.3, the entity pair in the given context may be separated by a large gap. To match both entities using a single tree structure is hard to achieve, since large structure is sparse in the data set. TSKs/aTSKs may address this issue by extracting features of non-contiguous subtree sequence. As a result, TSKs/aTSKs, which captures patterns across both entities may benefit those tasks with anchored text such as relation extraction.

Unlike relation extraction, question classification may consider non-contiguous

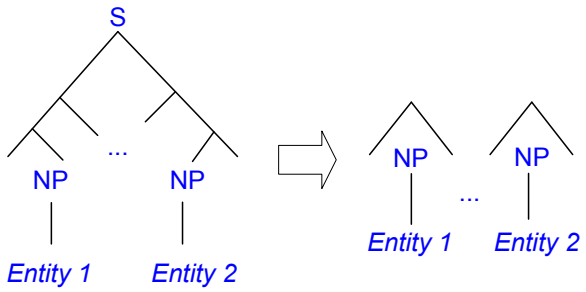


Figure 5.3: Non-contiguous tree sequence features

patterns to be less important, since it is sometimes unnecessary to deeply explore the non-contiguous patterns for certain question sentences. For example, the question sentences beginning with single inquiry words "where" and "when" are easy to be correctly categorized. On the other hand, the improvement on classification accuracy still suggests that some question sentences indeed receive benefit from the non-contiguous tree sequence features obtained from TSKs.

In addition, one of the goals to design aTSKs is to employ the structure features over the context around the anchored structures. Unfortunately, we cannot detect any improvement of MCT against PT for the corresponding TSKs/aTSKs in relation extraction. This finding seemingly suggests that the context information in MCT still may not be well utilized by the proposed TSKs/aTSKs. On the other hand, Zhou et al. (2007) studied the necessary tree spans on a small sample of ACE 2003 corpus and classified the relation instances into five categories. They showed that only the "predicate-linked and others" type may benefit from the context information and this type only takes a small amount of relation instances (19 out of 100). Therefore, it is quite understandable that kernels on MCT underperform kernels on PT. It only benefits a few instances to use the context information, while possibly bringing noisy features for most of other instances. Consequently, a universal model for different types of relation instances may not well facilitate those small amount of instances. In addition, the MCT structure may still contain some noisy structures, which may compromise the performance.

5.4 Summary

In this chapter, we applied the proposed tree sequence kernels (cTSK/TSKs/aTSKs) on two NLP applications, i.e. question classification and relation extraction. The key characteristic of the proposed cTSK and TSKs is that the captured structure features are enlarged from the structure of a single tree to the structure of multiple

trees (tree sequence). Experimental results on question classification and relation extraction suggest that the proposed cTSK/TSKs outperform the single tree based kernels (TK/PTK) in both applications, which verifies the advantages of the additional tree sequence features. Specifically, the structure of tree sequence more facilitates the task that focuses on disconnected constituents modeling, i.e. relation extraction. Leveraging TSKs, the proposed aTSKs are applied in relation extraction as well. By means of aTSKs, a further performance boosting is detected compared with the corresponding TSKs, which suggests the constraint of preserving the relative positions to the anchored entities for the tree sequence features is beneficial to relation extraction and it is effective of aTSKs to capture this kind of constraint. Experimental results also suggest that the extended constraints employed by $aTSK_1$s and $aTSK_2$s are too strict to capture the various effective structure features and the corresponding kernels cannot compete with aTSKs which have less constraint in relation extraction.

Chapter 6

A Kernel based Statistical Model for Bilingual Subtree Alignment

In this Chapter, the effectiveness of kernels over multiple trees will be verified on bilingual applications, where the input data are bilingual parse trees. Consequently, we name the corresponding kernel as Bilingual Tree (Sequence) Kernels (BTKs/BTSKs). The specific application that the kernels are applied in is bilingual subtree alignment. To apply BTKs/BTSKs in this task, we propose a kernel based statistical alignment model. In this model, BTKs/BTSKs over a variety of feature spaces are proposed to capture the structural similarities across a pair of syntactic translational equivalences. Along with BTKs/BTSKs, various lexical and syntactic structural features are proposed to capture the correspondence between bilingual subtrees using a polynomial kernel. We then attempt to combine the polynomial kernel and BTKs/BTSKs to construct a composite kernel. The bilingual subtree alignment task is considered as a binary classification problem. We employ a kernel based classifier with the composite kernel to classify each candidate of subtree pair as aligned or unaligned. Then greedy search is performed according to the definition of subtree alignment within the space of candidates classified as aligned.

This chapter is organized as follows: In Section 6.1, the task of bilingual subtree alignment is formally defined, followed by an introduction of the background and the related work of the task. In Section 6.3, we design several bilingual tree (sequence) kernel feature spaces. In Section 6.4, some heuristic features are proposed. In Section 6.5, we propose the kernel based alignment model which combines the feature based polynomial kernel and BTKs/BTSKs over parse trees as composite kernels. Experiments are carried out in two folds. First, the subtree alignment results are evaluated. Second, the aligned subtrees from the best aligner are employed as translation rules in the state-of-the-art machine translation systems and the BLEU score (Papineni et al., 2002) (a translation metric) is measured for the translation quality. These results are presented in Section 6.6 and Section 6.7 followed by the summary of this chapter.

6.1 Task Definition

A subtree alignment process pairs up the subtrees across bilingual parse trees, whose lexical leaf nodes covered are translational equivalent, i.e. sharing the same semantics. Grammatically, the task conducts links between syntactic constituents with the maximum tree structures generated over their word sequences in bilingual tree pairs.

In general, subtree alignment can also be interpreted as conducting multiple links across internal nodes between sentence-aligned tree pairs as shown in Fig. 6.1. The aligned subtree pairs usually maintain a non-isomorphic relation with each other especially for higher layers. We utilize the same criteria as Tinsley et al. (2007) in our study of subtree alignment as follows:

- a node can only be linked once;
- descendants of a source linked node may only link to descendants of its target

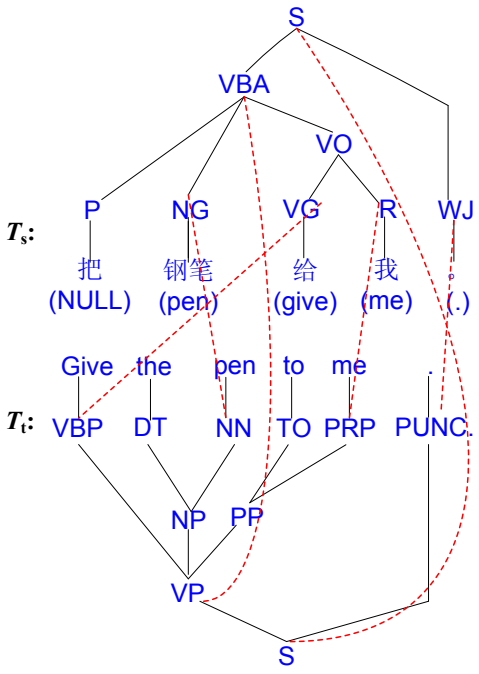


Figure 6.1: Subtree alignment as referred to Node alignment

linked counterpart;

- ancestors of a source linked node may only link to ancestors of its target linked counterpart.

where the term "node" refers to the root of a subtree, which can be used as a representative of the subtree.

6.2 Background and Related Work

Recent research in Statistical Machine Translation (SMT) tends to incorporate more linguistically grammatical information into the translation model known as linguistically motivated syntax-based models. To develop such models, the phrasal structure parse tree is usually adopted as the representation of bilingual sentence pairs either on the source side (Huang, Knight, and Joshi, 2006; Liu, Liu, and Lin,

2006) or on the target side (Galley et al., 2006; Marcu et al., 2006), or even on both sides (Graehl and Knight, 2004; Zhang et al., 2007). Most of the above models either construct a pipeline to transform from/to tree structure, or synchronously generate two trees in parallel (i.e., synchronous parsing). Both cases require syntactically rich translational equivalences to handle non-local reordering. However, most current works obtain the syntactic translational equivalences by initially conducting alignment on the word level. To employ word alignment as a hard constraint for rule extraction has difficulty in capturing such non-local phenomena and will fully propagate the word alignment error to the later stage of rule extraction.

Alternatively, some initial attempts have been made to directly conduct syntactic structure alignment. As mentioned in Tinsley et al. (2007), the earliest work usually constructs the structure alignment by hand, which is time-consuming. Recent research tries to automatically align the bilingual syntactic subtrees.

Early approaches pay much attention to conduct alignment for dependency structures. Among them, Matsumoto, Ishimoto, and Utsuro (1993), Meyers, Yangarber, and Grishman (1996), Yamamoto and Matsumoto (2000) propose methods mainly for obtaining bilingual lexical knowledge or producing translation templates for EBMT by heuristically performing some similarity computation techniques. Watanabe, Kurohashi, and Aramaki (2000) describe a method which initially ensures word correspondences and then confirms phrasal correspondences based on word correspondences. Menezes and Richardson (2001) develop a "best-first" alignment method which iteratively applies the lexical alignment rules and transfers mapping rules on a relation based tree structure. Eisner (2003) proposes a tree-mapping (tree-to-tree) framework targeting the unique translation strategy named Statistical Tree Substitution Grammar (STSG) in which the mapping items in terms of elementary trees are not subtrees but subgraphs with frontier nodes to be expanded.

As for the work in the phrase structure trees, Imamura (2001) presents a hierarchical phrase alignment method with constituent based parsing. However, the heuristics defined in this work are tightly dependent on word alignment and only allow the spans with identical grammar node categories to be aligned regardless of the translational structure divergence across languages. Gildea (2003) proposes a method for word to word alignment by aligning non-isomorphic phrase-structure trees using a stochastic clone operation which allows the modification of the tree structure. Groves, Hearne, and Way (2004) present a rule-based aligner by means of prior conducted word alignments. Their algorithm proceeds from the aligned lexical terminal nodes in a bottom-up fashion, performing a best-first search with heuristics.

However, most of these works suffer from the following problems. Firstly, the alignment is conducted based on heuristic rules, which may lose extensibility and generality in spite of accommodating some common cases. Secondly, various similarity computation methods are used based merely on lexical translation probabilities regardless of the structural features. This may be due to the fact that the syntactic structures in a parse tree pair are hard to describe using plain features. In addition, explicitly utilizing syntactic tree fragments results in exponentially high dimensional feature vectors, which is hard to compute. We believe the structure information is an important issue to capture the non-local structural divergence of languages by modeling beyond the plain text. In order to utilize the structure features embedded in the parse trees, we will apply BTKs/BTSKs on the subtree alignment task to explore the syntactic structure features.

6.3 Structure Space for BT(S)Ks

The syntactic translational equivalences under BTKs/BTSKs are evaluated with respect to the substructures factorized from the candidate subtree pairs. In this

section, we propose different substructures to facilitate the measurement of syntactic similarity for subtree alignment and illustrate those substructures on BTKs. Although the substructures are only illustrated for BTKs, some of the substructures can be explored by BTSKs as well and it is quite obvious to extend those substructures from single tree (BTKs) to tree sequence (BTSKs). In addition, since the proposed BTKs/BTSKs can be computed by individually evaluating the source and target monolingual tree kernels, the definition of the substructure can be simplified to base only on monolingual subtrees.

Subset Tree

Subset Tree (SST) is the substructure explored by Collins and Duffy's tree kernel. An SST is any subgraph, which includes more than one non-terminal node, with the constraint that the entire rule productions are included. Fig. 6.2 shows an example of the SSTs decomposed from the source subtree rooted at VP*.

Root directed Subset Tree

Collins and Duffy's tree kernel achieves decent performance using the SSTs due to the rich exploration of syntactic information. However, the bilingual subtree alignment task requires strong capability of discriminating the subtrees with their roots across adjacent generations, because those candidates share many identical SSTs. As illustrated in Fig. 6.2, the source subtree rooted at VP*, which should be aligned to the target subtree rooted at NP*, may be likely aligned to the subtree rooted at PP*, which shares quite a similar context with NP*. It is also easy to show that the latter shares all the SSTs that the former obtains. In consequence, the values of the SST based kernel function are quite similar between the candidate subtree pair rooted at (VP*,NP*) and (VP*,PP*).

In order to effectively differentiate the candidates like the above, we propose the Root directed Subset Tree (RdSST) by encapsulating each SST with the root of the given subtree. As shown in Fig. 6.2, a substructure is considered identical to

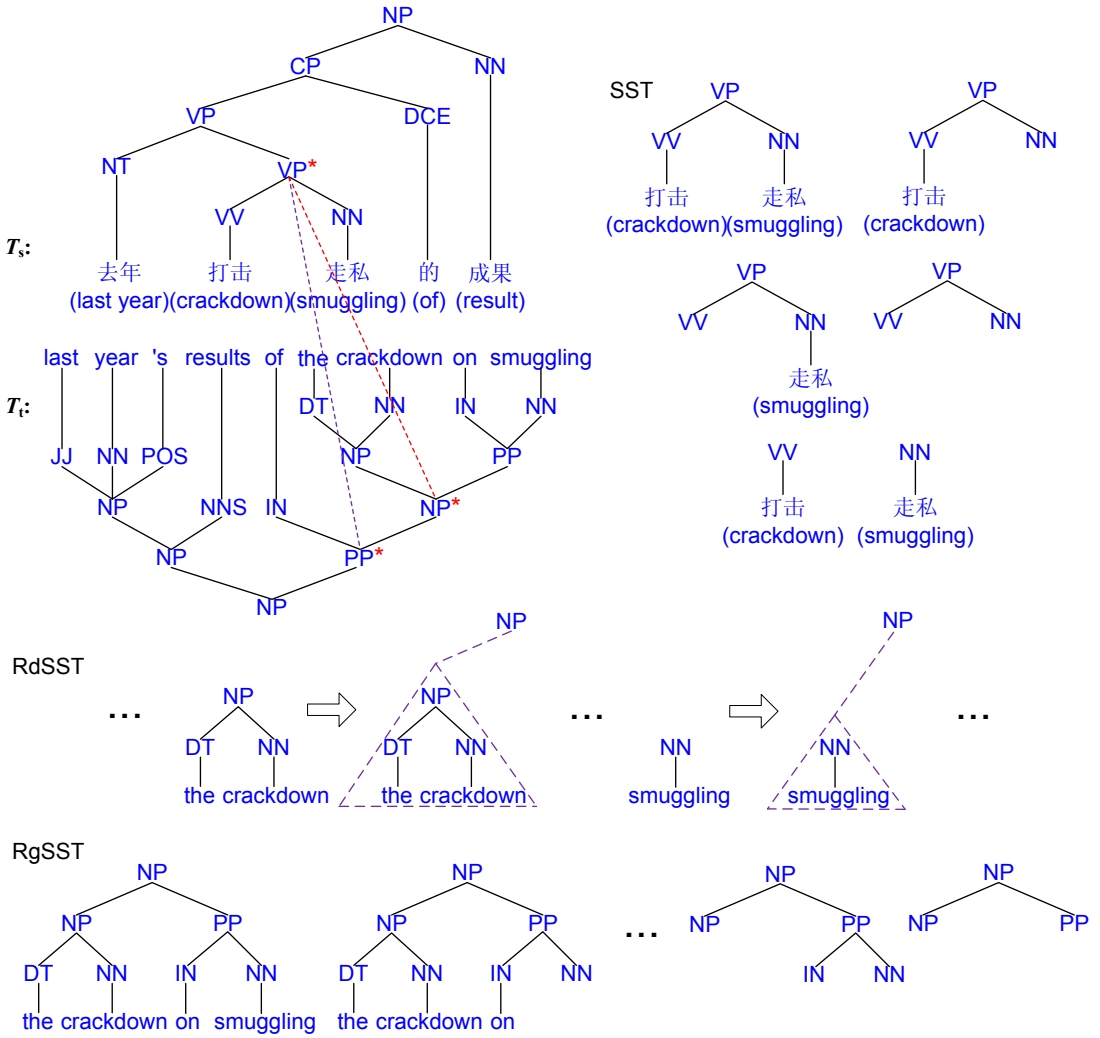


Figure 6.2: Illustration of SST, RdSST and RgSST

the given examples, when the SST is identical and the root tag of the given subtree is NP. As a result, the kernel function (Eq. 3.21) can be redefined as:

$$\begin{aligned} K_t(T, T') &= \sum_{n \in N_T} \sum_{n' \in N'_T} \Lambda(n, n') I(r, r') \\ &= I(r, r') \sum_{n \in N_T} \sum_{n' \in N'_T} \Lambda(n, n') \end{aligned} \tag{6.1}$$

where r and r' are the root nodes of the subtree T and T' respectively. The indicator function $I(r, r')$ equals to 1 if r and r' are identical, and 0 otherwise. Although

defined for individual SST, the indicator function can be evaluated outside the summation, without increasing the computational complexity of the kernel function.

Root generated Subset Tree

Some grammatical tags (NP/VP) may have identical tags as their parents or children which may make RdSST less effective. Consequently, we step further to propose the substructure of Root generated Subset Tree (RgSST). An RgSST requires the root node of the given subtree to be part of the substructure. In other words, all substructures should be generated from the root of the given subtree as presented in Fig. 6.2. Therefore the kernel function can be simplified to only capture the substructure rooted at the root of the subtree.

$$K_t(T, T') = \Lambda(r, r') \tag{6.2}$$

where r and r' are the root nodes of the subtree T and T' respectively. The time complexity is reduced to $O(|N_S| + |N_{S'}| + |N_T| + |N_{T'}|)$.

Root only

More aggressively, we can simplify the kernel to only measure the common root node without considering the complex tree structures. Therefore the kernel function is simplified to be a binary function with time complexity $O(1)$.

$$K_t(T, T') = I(r, r') \tag{6.3}$$

Although those substructures are exemplified for BTKs, in fact SST and RdSST can be applied for BTSKs as well. When applying SST to BTSKs, the substructures explored are just common tree sequence structures. When applying RdSST to BTSKs, the substructures are the common tree sequences as well as a hard constraint to match the root nodes of the candidate subtree pairs.

6.4 Heuristic Feature Functions

Besides kernels over parse trees, we introduce various heuristic lexical and structural feature functions. The lexical features with directions are defined as conditional feature functions based on the conditional lexical translation probabilities. The plain syntactic structural features can deal with the structural divergence of bilingual parse trees in a more general perspective.

6.4.1 Lexical and Word Alignment Features

In this section, we define seven lexical features to measure semantic similarity of a given subtree pair.

Internal Lexical Features

We define two lexical features with respective to the internal span of the subtree pair.

$$\phi_1(S|T) = \left(\prod_{v \in in(T)} \sum_{u \in in(S)} P(u|v) \right)^{\frac{1}{|in(T)|}} \tag{6.4}$$

$$\phi_2(T|S) = \left(\prod_{u \in in(S)} \sum_{v \in in(T)} P(v|u) \right)^{\frac{1}{|in(S)|}} \tag{6.5}$$

where $P(v|u)$ refers to the lexical translation probability from the source word u to the target word v within the subtree spans, while $P(u|v)$ refers to that from target to source; $in(S)$ refers to the word set for the internal span of the source subtree S, while $in(T)$ refers to that of the target subtree T.

Internal-External Lexical Features

These features are motivated by the fact that lexical translation probabilities within the translational equivalence tend to be high, and that of the non-equivalent

counterparts tend to be low.

$$\phi_3(S|T) = \left(\prod_{v \in in(T)} \sum_{u \in out(S)} P(u|v) \right)^{\frac{1}{|in(T)|}} \tag{6.6}$$

$$\phi_4(T|S) = \left(\prod_{u \in in(S)} \sum_{v \in out(T)} P(v|u) \right)^{\frac{1}{|in(S)|}} \tag{6.7}$$

where $out(S)$ refers to the word set for the external span of the source subtree S, while $out(T)$ refers to that of the target subtree T.

Internal Word Alignment Features

The word alignment links account much for the co-occurrence of the aligned terms. We define the internal word alignment features as follows:

$$\phi_5(S,T) = \frac{\sum_{v \in in(T)} \sum_{u \in in(S)} \delta(u,v) \cdot (P(u|v) \cdot P(v|u))^{\frac{1}{2}}}{(|in(S)| \cdot |in(T)|)^{\frac{1}{2}}} \tag{6.8}$$

where

$$\delta(u,v) = \begin{cases} 1 & \text{if } (u,v) \text{ is aligned} \\ 0 & \text{otherwise} \end{cases} \tag{6.9}$$

The binary function $\delta(u,v)$ is employed to trigger the computation only when a word aligned link exists for the two words (u,v) within the subtree span.

Internal-External Word Alignment Features

Similar to the lexical features, we also introduce the internal-external word alignment features as follows:

$$\phi_6(S,T) = \frac{\sum_{v \in in(T)} \sum_{u \in out(S)} \delta(u,v) \cdot (P(u|v) \cdot P(v|u))^{\frac{1}{2}}}{(|out(S)| \cdot |in(T)|)^{\frac{1}{2}}} \tag{6.10}$$

$$\phi_7(S,T) = \frac{\sum_{v \in out(T)} \sum_{u \in in(S)} \delta(u,v) \cdot (P(u|v) \cdot P(v|u))^{\frac{1}{2}}}{(|in(S)| \cdot |out(T)|)^{\frac{1}{2}}} \tag{6.11}$$

where

$$\delta(u,v) = \begin{cases} 1 & \text{if } (u,v) \text{ is aligned} \\ 0 & \text{otherwise} \end{cases} \tag{6.12}$$

6.4.2 Online Structural Features

In addition to the lexical correspondence, we also capture the structural divergence by introducing the following tree structural features.

Span difference

Translational equivalent subtree pairs tend to share similar length of spans. Thus the model will penalize the candidate subtree pairs with largely different length of spans.

$$\varphi_1(S,T) = \left| \frac{|in(S)|}{|in(\mathbf{S})|} - \frac{|in(T)|}{|in(\mathbf{T})|} \right| \tag{6.13}$$

S and **T** refer to the entire source and target parse trees respectively. Therefore, $|in(\mathbf{S})|$ and $|in(\mathbf{T})|$ are the respective span length of the parse tree used for normalization.

Number of Descendants

Similarly, the number of the root's descendants of the aligned subtrees should also correspond.

$$\varphi_2(S,T) = \left| \frac{|R(S)|}{|R(\mathbf{S})|} - \frac{|R(T)|}{R(|\mathbf{T})|} \right| \tag{6.14}$$

where $R(.)$ refers to the descendant set of the root to a subtree.

Tree Depth difference

Intuitively, translational equivalent subtree pairs tend to have similar depth from the root of the parse tree. We allow the model to penalize the candidate subtree pairs with quite different distance of path from the root of the parse tree to the root of the subtree.

$$\varphi_3(S,T) = \left| \frac{Depth(S)}{Height(\mathbf{S})} - \frac{Depth(T)}{Height(\mathbf{T})} \right| \tag{6.15}$$

6.5 The Alignment Model

Given feature spaces defined in the last two sections, we propose a two phase subtree alignment model as follows:

In the first phase, a kernel based classifier, SVM in our study, is employed to classify each candidate subtree pair as *aligned* or *unaligned*. The feature vector of the classifier is computed using a composite kernel:

$$K(S \cdot T, S' \cdot T') = \theta_0 \widehat{K}_p(S \cdot T, S' \cdot T') + \sum_{i=1}^{K} \theta_i \widehat{K}^i_{BT(S)K}(S \cdot T, S' \cdot T') \qquad (6.16)$$

$\widehat{K}_p(\cdot,\cdot)$ is the normalized form of the polynomial kennel $K_p(\cdot,\cdot)$, which is a polynomial kernel with the degree of 2, utilizing the plain features. $\widehat{K}^i_{BT(S)K}(\cdot,\cdot)$ is the normalized form of $K^i_{BT(S)K}(\cdot,\cdot)$, exploring the corresponding substructure space. The composite kernel can be constructed using the polynomial kernel for plain features and various BT(S)Ks for tree structure by linear combination with coefficient θ_i, where $\sum_{i=0}^{K} \theta_i = 1$.

In the second phase, we adopt a greedy search with respect to the alignment probabilities. Since SVM is a large margin based discriminative classifier rather than a probabilistic model, we introduce a sigmoid function to convert the distance against the hyperplane to a posterior alignment probability as follows:

$$P(a_+|S,T) = \frac{1}{1+e^{-D_+}} \qquad (6.17)$$

$$P(a_-|S,T) = \frac{1}{1+e^{-D_-}} \qquad (6.18)$$

where D_+ is the distance for the instances classified as *aligned* and D_- is that for the *unaligned*. We use $P(a_+|S,T)$ as the confidence to conduct the *sure* links for those classified as *aligned*. On this perspective, the alignment probability is suitable as a searching metric. The search space is reduced to that of the candidates classified as *aligned* after the first phase.

6.6 Experiments on Bilingual Subtree Alignment

In order to evaluate the effectiveness of the alignment model and its capability in the applications requiring syntactic translational equivalences, we employ two cor-

pora to carry out the subtree alignment evaluation. The first is HIT gold standard English Chinese parallel tree bank referred as HIT corpus[1]. The other is the automatically parsed bilingual tree pairs selected from FBIS corpus (allowing minor parsing errors) with human annotated subtree alignment.

6.6.1 Data Preparation

HIT corpus, which is collected from English learning text books in China as well as example sentences in dictionaries, is used for the gold standard corpus evaluation. The word segmentation, tokenization and parse tree in the corpus are manually constructed or checked. The corpus is constructed with manually annotated subtree alignment. The annotation strictly preserves the semantic equivalence of the aligned subtree pair. Only *sure* links are conducted in the internal node level, without considering *possible* links adopted in word alignment. A different annotation criterion of the Chinese parse tree, designed by the annotator, is employed. Compared with the widely used Penn TreeBank annotation, the new criterion utilizes some different grammar tags and is able to effectively describe some rare language phenomena in Chinese. The annotator still uses Penn TreeBank annotation on the English side. The statistics of HIT corpus used in our experiment is shown in Table 6.1. We use 5000 sentences for experiment and divide them into three parts, with $3k$ for training, $1k$ for testing and $1k$ for tuning the parameters of kernels and thresholds of pruning the negative instances.

Most linguistically motivated syntax based SMT systems require an automatic parser to perform the rule induction. Thus, it is important to evaluate the subtree alignment on the automatically parsed corpus with parsing errors. In addition, HIT corpus is not applicable for MT experiment due to the problems of domain

[1]HIT corpus is designed and constructed by HIT-MITLAB. http://mitlab.hit.edu.cn/index.php/resources.html

	Chinese	English
# of Sentence pair	5000	
Avg. Sentence Length	12.93	12.92
Avg. # of subtree	21.40	23.58
Avg. # of alignment	11.60	

Table 6.1: Corpus Statistics for HIT corpus

	Chinese	English
# of Sentence pair	300	
Avg. Sentence Length	16.94	20.81
Avg. # of subtree	28.97	34.39
Avg. # of alignment	17.07	

Table 6.2: Statistics of FBIS selected Corpus

divergence, annotation discrepancy (Chinese parse tree employs a different grammar from Penn Treebank annotations) and degree of tolerance for parsing errors.

Due to the above issues, we annotate a new data set to apply the subtree alignment in machine translation. We randomly select 300 bilingual sentence pairs from the Chinese-English FBIS corpus with the length less or equal to 30 in both the source and target sides. The selected plain sentence pairs are further parsed by Stanford parser (Klein and Manning, 2003) on both the English and Chinese sides. We manually annotate the subtree alignment for the automatically parsed tree pairs according to the definition in Section 6.1. To be fully consistent with the definition, we strictly preserve the semantic equivalence for the aligned subtrees to keep a high precision. In other words, we do not conduct any doubtful links. The corpus is further divided into 200 aligned tree pairs for training and 100 for testing as shown in Table 6.2.

6.6.2 Baseline approaches

We implement two baselines for comparison. The first baseline is the heuristic based method proposed in Tinsley et al. (2007). The second baseline is to apply the proposed heuristic feature functions in a Maximum Entropy (ME) model instead of a Kernel Machine like SVM (Sun, Zhang, and Tan, 2010a).

For the first baseline proposed by Tinsley et al. (2007), given a tree pair $< \mathbf{S}, \mathbf{T} >$, the baseline approach first takes all the links between the subtree pairs as alignment hypotheses, i.e., the Cartesian product of the two subtree sets:

$$\{S_1, \ldots, S_i, \ldots, S_I\} \times \{T_1, \ldots, T_j, \ldots, T_J\}$$

By using the lexical translation probabilities, each hypothesis is assigned an alignment score. All hypotheses with zero score are pruned out. Then the algorithm iteratively selects the link of the subtree pairs with the maximum score as a *sure* link, and blocks all hypotheses that contradict with this link and itself, until no non-blocked hypotheses remain.

The baseline system uses many heuristics in searching the optimal solutions with alternative score functions. Heuristic *skip1* skips the tied hypotheses with the same score, until it finds the highest scoring hypothesis with no competitors of the same score. Heuristic *skip2* deals with the same problem. Initially, it skips over the tied hypotheses. When a hypothesis subtree pair (S_i, T_j) without any competitor of the same score is found, where neither S_i nor T_j has been skipped over, the hypothesis is chosen as a *sure* link. Heuristic *span1* postpones the selection of the hypotheses on the POS level. Since the highest scoring hypotheses tend to appear on the leaf nodes, it may introduce ambiguity when conducting the alignment for a POS node whose child word appears twice in a sentence.

The baseline method proposes two score functions based on the lexical translation probability. They also compute the score function by splitting the tree into the internal and external components.

Tinsley et al. (2007) adopt the lexical translation probabilities dumped by GIZA++ (Och and Ney, 2003) to compute the span based scores for each pair of subtrees. Although all of their heuristics combinations are reimplemented in our study, we only present the best result among them with the highest Recall and F-value as our baseline, denoted as *skip2_s1_span1*[2].

As for the baseline using the ME model, we employ all the lexical $\phi_{1\sim7}$ and structural $\varphi_{1\sim3}$ feature functions. In addition, we adopt the combination of root grammar tags of the subtree pairs as binary features. For more details of this model, readers may refer to Sun, Zhang, and Tan (2010a).

6.6.3 Experimental Settings

We use SVM with binary classes as the classifier. In case of the implementation, we modify the Tree Kernel tool (Moschitti, 2006) and SVMLight (Joachims, 1999). The coefficient θ_i for the composite kernel are tuned with respect to F-measure (**F**) on the development set of HIT corpus. We empirically set $C = 2.4$ for SVM and and use the default parameter $\lambda = 0.4$ for BTKs.

As for the tree sequence kernel, we only employ the contiguous tree sequence kernel (cTSK) for this task, namely Bilingual contiguous Tree Sequence Kernels (BcTSKs). The experiments are carried out on the same HIT corpus as BTKs. The parameters of λ, μ for cTSK and the coefficient θ_i for the composite kernel are tuned on the development set of HIT corpus.

Since the negative training instances largely overwhelm the positive instances, we prune the negative instances using the thresholds according to the lexical feature functions $\phi_{1\sim4}$ and online structural feature functions $\varphi_{1\sim3}$. Those thresholds are also tuned on the development set of HIT corpus with respect to F-measure.

[2] *s1* denotes score function 1 in (Tinsley et al., 2007), *skip2_s1_span1* denotes the utilization of heuristics *skip2* and *span1* while using score function 1

Feature Space	**P**	**R**	**F**
Lex	61.62	58.33	59.93
Lex+Online Str	70.08	69.02	69.54
Plain[3]+dBTK-SST	80.36	78.08	79.20
Plain+dBTK-RdSST	87.52	74.13	80.27
Plain+dBTK-RgSST	88.54	70.18	78.30
Plain+dBTK-Root	81.05	84.38	**82.68**
Plain+dBcTSK-SST	84.59	75.62	79.86
Plain+dBcTSK-RdSST	84.14	79.13	81.56
Plain+iBTK-SST	81.57	73.51	77.33
Plain+iBTK-RdSST	82.27	77.58	80.00
Plain+iBTK-RgSST	82.92	78.77	**80.80**
Plain+iBTK-Root	76.37	76.81	76.59
Plain+iBcTSK-SST	81.44	78.80	80.10
Plain+iBcTSK-RdSST	82.21	79.81	**80.99**
Plain+dBTK-Root+iBTK-RgSST	85.53	85.21	**85.32**
Baseline (Tinsley et al., 2007)	64.14	66.99	65.53
Baseline(Plain) (Sun, Zhang, and Tan, 2010a)	57.64	63.11	60.25
Baseline(Plain+Binary feature) (Sun, Zhang, and Tan, 2010a)	73.14	85.11	78.67

Table 6.3: Structure feature contribution for HIT test set

To learn the lexical and word alignment features for both the proposed model and the baseline method, we train GIZA++ on the entire FBIS bilingual corpus with $240k$ sentence pairs. The evaluation is conducted by means of Precision (**P**), Recall (**R**) and F-measure (**F**).

6.6.4 Experimental Results

In Tables 6.3 and 6.4, we incrementally enlarge the feature spaces in certain order for both corpora and examine the feature contribution to the alignment results. In detail, the iBTKs/iBcTSKs and dBTKs/dBcTSKs are firstly combined with the polynomial kernel for plain features individually, then the best iBTK and dBTK

[3]Plain=Lex+Online Str

Feature Space	**P**	**R**	**F**
Lex	73.48	71.66	72.56
Lex+Online Str	77.02	73.63	75.28
Plain+dBTK-SST	81.44	74.42	77.77
Plain+dBTK-RdSST	81.40	69.29	74.86
Plain+dBTK-RgSST	81.90	67.32	73.90
Plain+dBTK-Root	78.60	80.90	**79.73**
Plain+iBTK-SST	82.94	79.44	81.15
Plain+iBTK-RdSST	83.14	80.00	**81.54**
Plain+iBTK-RgSST	83.09	79.72	81.37
Plain+iBTK-Root	78.61	79.49	79.05
Plain+dBTK-Root+iBTK-RgSST	82.70	82.70	**82.70**
Baseline (Tinsley et al., 2007)	70.84	78.70	74.36
Baseline(Plain) (Sun, Zhang, and Tan, 2010a)	72.03	80.95	76.23
Baseline(Plain+Binary feature) (Sun, Zhang, and Tan, 2010a)	76.08	85.29	80.42

Table 6.4: Structure feature contribution for FBIS test set

are chosen to construct a more complex composite kernel along with the polynomial kernel for both corpora. The experimental results show that:

- All the settings with structural features of the proposed approach achieve better performance than the first baseline (Tinsley et al., 2007), since the baseline method only assesses semantic similarity using the lexical features. In addition, most proposed composite kernels also outperform the other baseline (Sun, Zhang, and Tan, 2010a). The improvement suggests that the proposed framework with syntactic structural features is more effective in modeling the bilingual syntactic correspondence.

- By introducing BTKs to construct a composite kernel, the performance in both corpora is significantly improved against only using the polynomial kernel for plain features. This suggests that the structural features captured by BTKs are quite useful for the subtree alignment task. We also try to use

BTKs alone without the polynomial kernel for plain features; however, the performance is rather low. This suggests that the structure correspondence cannot be used to measure the semantically equivalent tree structures alone, since the same syntactic structure tends to be reused in the same parse tree and lose the ability of disambiguation to some extent. In other words, to capture the semantic similarity, structure features requires lexical features to cooperate.

- After comparing iBTKs with the corresponding dBTKs, we find that for *FBIS* corpus, iBTK greatly outperforms dBTK in any feature space except the Root space. However, when it comes the *HIT* corpus, the gaps between the corresponding iBTKs and dBTKs are much closer, while on the Root space, dBTK outperforms iBTK to a large amount. This finding can be explained by the relationship between the amount of training data and the high dimensional feature space. Since dBTKs are constructed in a joint manner which obtains a much larger high dimensional feature space than those of iBTKs, dBTKs require more training data to excel its capability, otherwise it will suffer from the data sparseness problem. The reason that dBTK outperforms iBTK in the feature space of Root in *FBIS* corpus is that although it is a joint feature space, the Root node pairs can be constructed from a close set of grammar tags and to form a relatively low dimensional space.

  As a result, when applying to *FBIS* corpus, which only contains limited amount of training data, dBTKs will suffer more from the data sparseness problem, and therefore, a relatively low performance. When enlarging the amount of training corpus to the *HIT* corpus, the ability of dBTKs excels and the benefit from data increasing of dBTKs is more significant than iBTKs.

- We also find that the introduction of BTKs gains more improvement in HIT gold standard corpus than in FBIS corpus. Other than the factor of the amount of training data, this is also because the plain features in Table 6.3 are not as effective as those in Table 6.4, since they are trained on FBIS corpus which facilitates Table 6.4 more with respect to the domains. On the other hand, the grammatical tags and syntactic tree structures are more accurate in HIT corpus, which facilitates the performance of BTKs in Table 6.3.

- On the comparison across the different feature spaces of BTKs, we find that SST, RdSST and RgSST are rather selective, since Recalls of those feature spaces are relatively low, exp. for HIT corpus. However, the Root substructure obtains a satisfactory Recall for both corpora. That's why we attempt to construct a more complex composite kernel in adoption of the kernel of dBTK-Root as below.

- As for the comparison of BTKs and BcTSKs, we detect improvement from BcTSKs against the corresponding BTKs, but the improvement is inconsistent. For the dependent kernels, Fscore for dBcTSK-SST and dBcTSK-RdSST improves by +0.66 and +1.29 against dBTK-SST and dBTK-RdSST respectively. For the independent kernels, Fscore for iBcTSK-SST improves by a large margin (+2.77) while for iBcTSK-RdSST, Fscore gains very little (+0.19). The above results reveal that although we detect improvement from introducing the contiguous subtree sequence features, the improvement is not so attractive. In fact, the best dependent kernel, i.e. dBTK-Root, is still single subtree based, while the best independent kernel, i.e. iBcTSK-RdSST, only benefits from little gain (+0.19) against iBTK-RdSST.

- To gain an extra performance boosting, we further construct a composite kernel which includes the best iBTK and the best dBTK for each corpus

along with the polynomial kernel for plain features. In the HIT corpus, we use dBTK in the Root space and iBTK in the RgSST space; while for FBIS corpus, we use dBTK in the Root space and iBTK in the RdSST space. The experimental results suggest that by combining iBTK and dBTK together, we can achieve more improvement.

6.7 Experiments on Machine Translation

In addition to the intrinsic alignment evaluation, we further conduct the extrinsic MT evaluation. We explore the effectiveness of subtree alignment for both phrase based and linguistically motivated syntax based SMT systems.

6.7.1 Experimental Settings

In the experiments, we train the translation model on FBIS corpus ($7.2M$ (Chinese) + $9.2M$ (English) words in $240,000$ sentence pairs) and train a 4-gram language model on the Xinhua portion of the English Gigaword corpus ($181M$ words) using the SRILM Toolkit (Stolcke, 2002). We use these sentences with less than 50 characters from the NIST MT-2002 test set as the development set (to speed up tuning for syntax based system) and the NIST MT-2005 test set as our test set. We use the Stanford parser (Klein and Manning, 2003) to parse bilingual sentences on the training set and Chinese sentences on the development and test set. The evaluation metric is case-sensitive BLEU-4.

For the phrase based system, we use Moses (Koehn et al., 2007) with its default settings. For the syntax based system, since subtree alignment can directly benefit Tree-to-Tree based systems, we apply the subtree alignment in a syntax system based on Synchronous Tree Substitution Grammar (STSG) (Zhang et al., 2007).

The STSG based decoder uses a pair of *elementary tree*[4] as a basic translation unit. Recent research on tree based systems shows that relaxing the restriction from tree structure to tree sequence structure (Synchronous Tree Sequence Substitution Grammar: STSSG) significantly improves the translation performance (Zhang et al., 2008b). We implement the STSG/STSSG based model in the Pisces decoder with the identical features and settings in Sun, Zhang, and Tan (2009). In the Pisces decoder, the STSSG based decoder translates each span iteratively in a bottom up manner which guarantees that when translating a source span, any of its subspans is already translated. The STSG based decoding can be easily performed with the STSSG decoder by restricting the translation rule set to be elementary tree pairs only.

As for the alignment setting, we use the word alignment trained on the entire FBIS ($240k$) corpus by GIZA++ with heuristic grow-diag-final for both Moses and the syntax system. For subtree alignment, we use the above word alignment to learn lexical/word alignment feature, and train with the FBIS training corpus (200) using the composite kernel of Plain+dBTK-Root+iBTK-RdSST.

6.7.2 Experimental Results

Compared with the adoption of word alignment, translational equivalences generated from structural alignment tend to be more grammatically aware and syntactically meaningful. However, utilizing syntactic translational equivalences alone for machine translation loses the capability of modeling non-syntactic phrases to some extent (Koehn, Och, and Marcu, 2003). Consequently, instead of using phrases constraint by subtree alignment alone, we attempt to combine word alignment and subtree alignment and deploy the capability of both with two methods.

[4] An elementary tree is a fragment whose leaf nodes can be either non-terminal symbols or terminal symbols.

System	Model	BLEU
Moses	BP	23.86
	DirC	23.98
	EWoS	24.48
Syntax STSG	STSG	24.71
	DirC	25.16
	EWoS	25.38
Syntax STSSG	STSSG	25.92
	DirC	25.95
	EWoS	26.45

Table 6.5: MT evaluation on various systems

- *Directly Concatenate* (DirC) is operated by directly concatenating the rule set generated from subtree alignment and the original rule set generated from word alignment (Tinsley, Hearne, and Way, 2009). As shown in Table 6.5, we gain minor improvement in the Bleu score for all configurations.

- Alternatively, we proposed a new approach to generate the rule set from the scratch. We constrain the bilingual phrases to be consistent with *Either Word alignment or Subtree alignment* (EWoS) instead of being originally consistent with the word alignment only. The method helps tailoring the rule set decently without redundant counts for syntactic rules. The performance is further improved compared to DirC in all systems.

The findings suggest that with the modeling of non-syntactic phrases maintained, more emphasis on syntactic phrases can benefit both the phrase and syntax based SMT systems.

To benefit intuitive understanding, we provide two alignment snippets in the MT training corpus in Fig. 6.3, where the red lines across the non-terminal nodes are the subtree aligned links conducted by our model, while the purple lines across the terminal nodes are the word alignment links trained by GIZA++. In the first example, “Israel” is wrongly aligned to two words in the source sentence by

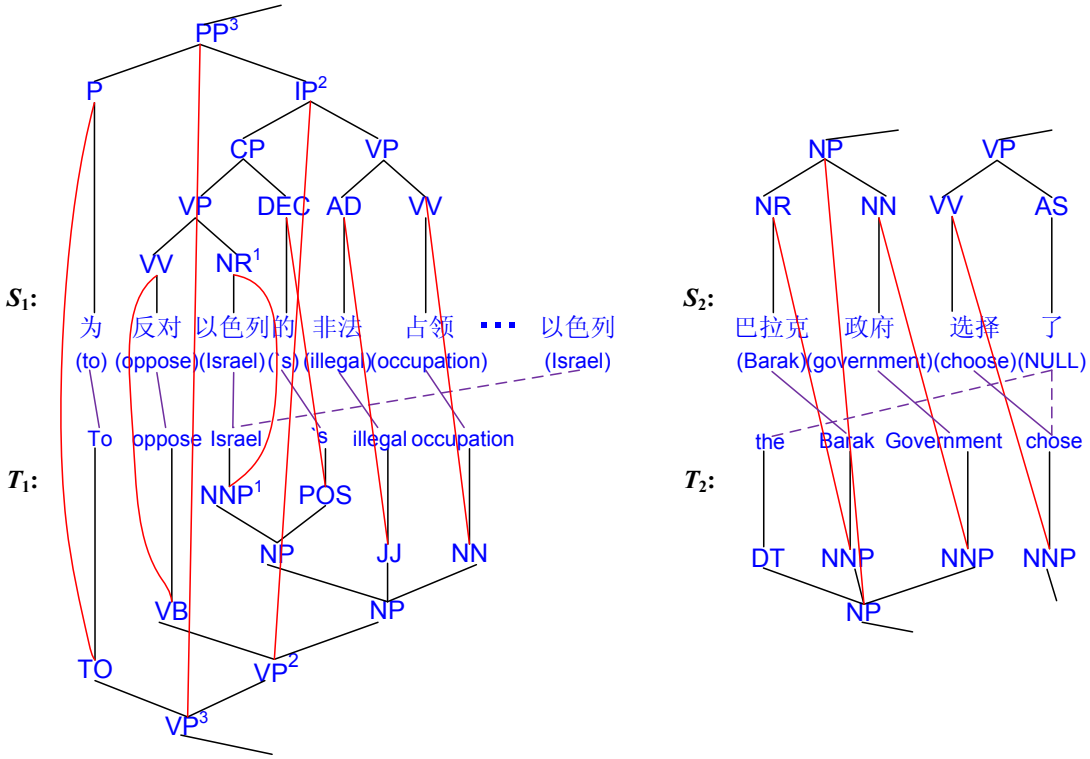


Figure 6.3: Comparison between the subtree alignment results and the word alignment results

GIZA++, where the wrong link is denoted by the dash line. This is common, since in a compound sentence in English, the entities appeared more than once are often replaced by pronouns at its later appearances. Therefore, the syntactic rules constraint by NR^1-NNP^1, IP^2-VP^2 and PP^3-VP^3 respectively cannot be extracted for syntax systems; while for phrase systems, context around the first "Israel" in the source sentence cannot be fully explored.

In the second example, the empty word "NULL" is wrongly aligned, which usually occurs in Chinese-English word alignment. As shown in Fig. 6.3, both cases can be resolved by subtree alignment conducted by our model, indicating that subtree alignment is a decent supplement to the word alignment rule set.

6.8 Summary

In this chapter, we explore syntactic structure features by means of Bilingual Tree (Sequence) Kernels (BTKs/BTSKs) and apply them to bilingual subtree alignment along with various lexical and plain structural features. We use both gold standard tree bank and the automatically parsed corpus for the subtree alignment evaluation. Experimental results show that our model significantly outperforms the two baseline methods and the proposed Bilingual Tree (Sequence) Kernels over various feature spaces are very effective in capturing the cross-lingual structural similarity. However, we only detect little improvement by enlarging the feature space from bilingual tree structures (BTK) to bilingual contiguous tree sequence structures (BcTSK). Further experiment shows that the obtained subtree alignment from the proposed subtree aligner benefits both phrase and syntax based MT systems by delivering more weight on syntactic phrases.

Chapter 7

Conclusion

7.1 Summary of Achievements

In this dissertation, a series of *tree sequence* based kernels are proposed to explore the structure features embedded in the phrase parse tree for NLP applications. In addition to the traditionally exploited connected graph based features, i.e. subtree, this dissertation systematically explores the disconnected structure features, i.e. subtree sequence, by means of kernels. In terms of this, this dissertation successfully provides some novel perspectives of structure features for NLP applications. Specifically, this dissertation achieves

- to propose contiguous Tree Sequence Kernel (cTSK) which is able to capture the structure features of a contiguous subtree sequence, and to propose an efficient algorithm to evaluate the kernel function.

- to propose Tree Sequence Kernels (TSKs) which are able to capture the structure of a subtree sequence, both contiguous and non-contiguous, and to propose various efficient algorithms to evaluate the kernel functions based on different structure weighting schemes.

- to propose Anchored Tree Sequence Kernel (aTSK) which is able to capture the spatial relationship between the subtree sequence features and the target anchored structures.

- to extend the tree (sequence) based kernels from the single parse tree to multiple parse trees and explore the structure features across multiple trees both dependently and independently.

- to apply the proposed kernels to various NLP applications, i.e. Question Classification, Relation Extraction and Bilingual Subtree Alignment, and conduct comparative experiments to demonstrate the effectiveness of these kernels.

- to propose a generic framework which employs a kernel based statistical alignment model for the task of Bilingual Subtree Alignment and integrate the tree (sequence) based kernels on bilingual parse trees in this framework along with various heuristic feature functions.

Specifically, Chapter 4 proposes a series of kernels constructed on the different tree sequence feature spaces. Contiguous Tree Sequence Kernel (cTSK) can explore the structure features of the contiguous subtree sequences. In order to evaluate cTSK, this thesis proposes an efficient algorithm to limit the time complexity to be the same as tree kernel of $O(|N| \cdot |N'|)$, with respect to the number of nodes in the parse trees. In addition, this thesis proposes Tree Sequence Kernels (TSKs). Compared with cTSK, TSKs further extends the feature space by allowing non-contiguous subtree sequence features. The generalized form of TSK is verified to be a valid kernel by constructing it based on the mapping kernel paradigm. Three approaches of penalizing the substructures are proposed. In order to efficiently evaluate TSKs, Set Sequence Kernels (SSKs) are proposed, which help capture the root node sequence of the subtree sequence. All TSKs can be evaluated within $O(m|N| \cdot |N'|)$, where m is the maximal subtrees allowed in a subtree sequence.

Based on the TSKs, this thesis further proposes Anchored Tree Sequence Kernels (aTSKs) to model the relationship across the anchored structures embedded in the parse tree. aTSKs can directly correspond the structure features to the anchored structures. Efficient algorithms for aTSKs are derived based on TSKs. By more aggressively constraining the feature space, this thesis proposes $aTSK_1$ and $aTSK_2$ and verify that $aTSK_1$ is a valid kernel while $aTSK_2$ is not. Finally, this thesis proposes to extend the tree (sequence) based kernels from the single parse tree to multiple parse trees to achieve multilingual tree (sequence) kernels.

In Chapter 5, to verify the effectiveness of the proposed kernels on the single parse tree, this thesis applies the proposed kernels in two NLP applications, i.e. Question Classification and Relation Extraction. Experimental results suggest that TSKs significantly outperform Collins and Duffy's tree kernel and Partial Tree Kernel (PTK) for both tasks, which reveals that the structure features expressed as a sequence of subtrees are effective and play a complementary role to the single subtree. In addition, compared with the task of question classification, relation extraction gains more improvement by using TSKs. It can be attributed to the reason that relation extraction is a task constructed on multiple items, which are highly likely to be disjointed. Therefore, TSKs that capture patterns across non-contiguous context can perform decently on relation extraction. Moreover, aTSKs, which restrict the syntactic features to directly cover the target entities, significantly outperform TSKs in relation extraction. This indicates that noisy structure features can be effectively pruned by relating the tree sequence features to the relational candidate pairs. The above experimental results demonstrate that there is useful syntactic and semantic information embedded in the tree sequence structures, which can provide discriminative patterns for NLP tasks.

In chapter 6, the proposed tree (sequence) based kernels are applied in multiple parse trees by means of Bilingual Subtree Alignment. A kernel based statistical

alignment model is proposed for this task. In this model, Bilingual Tree (Sequence) Kernels (BTKs/BTSKs) are proposed and applied along with various lexical and plain structural features. Experimental results show that the proposed model significantly outperforms the two baseline methods and the proposed Bilingual Tree (Sequence) Kernels over various feature spaces are very effective in capturing the cross-lingual structural similarity. Further experiment shows that the obtained subtrees from the proposed subtree aligner benefit both phrase and syntax based SMT systems by delivering larger weights on syntactic phrases.

7.2 Future Directions

In view of the success of the tree sequence based kernels, it is worthwhile applying the proposed kernels in more applications. For example, tree kernel has been demonstrated to be able to improve parsing results by reranking the N-best list (Collins and Duffy, 2002). Hence, it will be good to see parsing results can be further improved by examining the tree sequence structures. In addition, aTSKs may benefit the tasks that require deep syntactic and semantic analysis for target candidates, such as Semantic Role Labeling (SRL). In SRL, argument classification aims to identify the relation of an argument-predicate pair which can be considered as the target constituents.

Additionally, it is worthwhile investigating what kinds of substructures contribute most to a particular type of testing instances. Further study on this problem will also inspire new approaches on the modification of parse tree structures.

Another promising direction is to extend the current well-designed single subtree based kernels by accommodating the subtree sequence features. As introduced, partial tree kernel (Moschitti, 2006) and grammar driven tree kernel (Zhang et al., 2008a) both have demonstrated their goodness in certain tasks. Hence, it is worthy of studying the effectiveness of the tree sequence features in these kernels.

References

[Bloehdorn and Moschitti2007] Bloehdorn, S. and A. Moschitti. 2007. Structure and semantics for expressive text kernels. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 861–864. ACM.

[Bunescu and Mooney2005] Bunescu, R.C. and R.J. Mooney. 2005. A shortest path dependency kernel for relation extraction. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 724–731. Association for Computational Linguistics.

[Cancedda et al.2003] Cancedda, N., E. Gaussier, C. Goutte, and J.M. Renders. 2003. Word sequence kernels. *The Journal of Machine Learning Research*, 3:1059–1082.

[Charniak2001] Charniak, E. 2001. Immediate-head parsing for language models. In *Proceedings of ACL*, pages 124–131.

[Collins1997] Collins, M. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics.

[Collins and Duffy2002] Collins, M. and N. Duffy. 2002. Convolution kernels for natural language. *Advances in neural information processing systems*, 1:625–632.

[Cortes and Vapnik1995] Cortes, C. and V. Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

[Culotta and Sorensen2004] Culotta, A. and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting*

of Association for Computational Linguistics, pages 423–430. Association for Computational Linguistics.

[Doddington et al.2004] Doddington, G., A. Mitchell, M. Przybocki, L. Ramshaw, S. Strassel, and R. Weischedel. 2004. The automatic content extraction (ace) program–tasks, data, and evaluation. In *Proceedings of LREC*, volume 4, pages 837–840. Citeseer.

[Eisner2003] Eisner, J. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of Association for Computational Linguistics-Volume 2*, pages 205–208. Association for Computational Linguistics.

[Galley et al.2006] Galley, M., J. Graehl, K. Knight, D. Marcu, S. DeNeefe, W. Wang, and I. Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 961–968. Association for Computational Linguistics.

[Gildea2003] Gildea, D. 2003. Loosely tree-based alignment for machine translation. In *Proceedings of the 41st Annual Meeting of Association for Computational Linguistics-Volume 1*, pages 80–87. Association for Computational Linguistics.

[Graehl and Knight2004] Graehl, J. and K. Knight. 2004. Training tree transducers. In *Proc. HLT-NAACL*, pages 105–112.

[Groves, Hearne, and Way2004] Groves, D., M. Hearne, and A. Way. 2004. Robust sub-sentential alignment of phrase-structure trees. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 1072–1079. Association for Computational Linguistics.

[Haasdonk2005] Haasdonk, B. 2005. Feature space interpretation of svms with

indefinite kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 482–492.

[Haussler1999] Haussler, D. 1999. Convolution kernels on discrete structures. *In Technical Report UCSCRL9910 UC*, 23(1):1–38.

[Herbrich2002] Herbrich, R. 2002. *Learning kernel classifiers: theory and algorithms*. The MIT Press.

[Hopcroft and Ullman1979] Hopcroft, J.E. and J.D. Ullman. 1979. *Introduction to automata theory, languages, and computation*, volume 3. Addison-wesley Reading, MA.

[Huang, Knight, and Joshi2006] Huang, L., K. Knight, and A. Joshi. 2006. A syntax-directed translator with extended domain of locality. In *Proceedings of the Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, pages 1–8. Association for Computational Linguistics.

[Imamura2001] Imamura, K. 2001. Hierarchical phrase alignment harmonized with parsing. In *Proceedings of the 6th Natural Language Processing Pacific Rim Symposium (NLPRS 2001)*, pages 377–384. Citeseer.

[Joachims1999] Joachims, T. 1999. Making large scale svm learning practical.

[Kambhatla2004] Kambhatla, N. 2004. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics.

[Kashima and Koyanagi2002] Kashima, H. and T. Koyanagi. 2002. Kernels for semi-structured data. In *Proceedings of the International Conference on Machine Learning*, pages 291–298. Citeseer.

[Kate2008] Kate, R.J. 2008. A dependency-based word subsequence kernel. In

Proceedings of the Conference on Empirical Methods in Natural Language Processing, pages 400–409. Association for Computational Linguistics.

[Klein and Manning2003] Klein, D. and C.D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics.

[Koehn et al.2007] Koehn, P., H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics.

[Koehn, Och, and Marcu2003] Koehn, P., F.J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics.

[Li and Roth2002] Li, X. and D. Roth. 2002. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

[Liu, Liu, and Lin2006] Liu, Y., Q. Liu, and S. Lin. 2006. Tree-to-string alignment template for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 609–616. Association for Computational Linguistics.

[Lodhi et al.2002] Lodhi, H., C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. 2002. Text classification using string kernels. *The Journal of Machine Learning Research*, 2:419–444.

[Marcu et al.2006] Marcu, D., W. Wang, A. Echihabi, and K. Knight. 2006. Spmt: Statistical machine translation with syntactified target language phrases. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 44–52. Association for Computational Linguistics.

[Matsumoto, Ishimoto, and Utsuro1993] Matsumoto, Y., H. Ishimoto, and T. Utsuro. 1993. Structural matching of parallel texts. In *Proceedings of the 31st Annual Meeting of Association for Computational Linguistics*, pages 23–30. Association for Computational Linguistics.

[Menezes and Richardson2001] Menezes, A. and S.D. Richardson. 2001. A best-first alignment algorithm for automatic extraction of transfer mappings from bilingual corpora. In *Proceedings of the workshop on Data-driven methods in machine translation-Volume 14*, pages 1–8. Association for Computational Linguistics.

[Meyers, Yangarber, and Grishman1996] Meyers, A., R. Yangarber, and R. Grishman. 1996. Alignment of shared forests for bilingual corpora. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 460–465. Association for Computational Linguistics.

[Miller et al.2000] Miller, S., H. Fox, L. Ramshaw, and R. Weischedel. 2000. A novel use of statistical parsing to extract information from text. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 226–233. Morgan Kaufmann Publishers Inc.

[Moschitti2004] Moschitti, A. 2004. A study on convolution kernels for shallow semantic parsing. In *Proceedings of the 42nd Annual Meeting of Association for Computational Linguistics*, pages 335–342. Association for Computational Linguistics.

[Moschitti2006] Moschitti, A. 2006. Efficient convolution kernels for dependency

and constituent syntactic trees. *Machine Learning: ECML 2006*, pages 318–329.

[Moschitti2008] Moschitti, A. 2008. Kernel methods, syntax and semantics for relational text categorization. In *Proceeding of the 17th ACM conference on Information and knowledge management*, pages 253–262. ACM.

[Moschitti, Pighin, and Basili2006a] Moschitti, A., D. Pighin, and R. Basili. 2006a. Semantic role labeling via tree kernel joint inference. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 61–68. Association for Computational Linguistics.

[Moschitti, Pighin, and Basili2006b] Moschitti, A., D. Pighin, and R. Basili. 2006b. Tree kernel engineering in semantic role labeling systems. In *Proceedings of the Workshop on Learning Structured Information in Natural Language Applications, EACL 2006*, pages 49–56.

[Moschitti, Pighin, and Basili2008] Moschitti, A., D. Pighin, and R. Basili. 2008. Tree kernels for semantic role labeling. *Computational Linguistics*, 34(2):193–224.

[Moschitti and Zanzotto2007] Moschitti, A. and F.M. Zanzotto. 2007. Fast and effective kernels for relational learning from texts. In *Proceedings of the 24th international conference on Machine learning*, pages 649–656. ACM.

[Moschitti and Zanzotto2008] Moschitti, A. and F.M. Zanzotto. 2008. Encoding tree pair-based graphs in learning algorithms: the textual entailment recognition case. In *22nd International Conference on Computational Linguistics*, pages 25–32.

[Nguyen, Moschitti, and Riccardi2009] Nguyen, T.V.T., A. Moschitti, and G. Riccardi. 2009. Convolution kernels on constituent, dependency and sequential structures for relation extraction. In *Proceedings of the 2009 Conference on*

Empirical Methods in Natural Language Processing, pages 1378–1387. Association for Computational Linguistics.

[Och and Ney2003] Och, F.J. and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51.

[Papineni et al.2002] Papineni, K., S. Roukos, T. Ward, and W.J. Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics.

[Radev et al.2005] Radev, D., W. Fan, H. Qi, H. Wu, and A. Grewal. 2005. Probabilistic question answering on the web. *Journal of the American Society for Information Science and Technology*, 56(6):571–583.

[Reichartz, Korte, and Paass2009] Reichartz, F., H. Korte, and G. Paass. 2009. Dependency tree kernels for relation extraction from natural language text. *Machine Learning and Knowledge Discovery in Databases*, pages 270–285.

[Rosenblatt1962] Rosenblatt, F. 1962. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*, volume 115. Spartan books Washington, DC.

[Schölkopf and Smola2002] Schölkopf, B. and A.J. Smola. 2002. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. the MIT Press.

[Shawe-Taylor and Cristianini2004] Shawe-Taylor, J. and N. Cristianini. 2004. *Kernel methods for pattern analysis*. Cambridge Univ Pr.

[Shen, Sarkar, and Joshi2003] Shen, L., A. Sarkar, and A.K. Joshi. 2003. Using ltag based features in parse reranking. In *Proceedings of the 2003 conference on Empirical methods in natural language processing-Volume 10*, pages 89–96. Association for Computational Linguistics.

[Shin and Kuboyama2008] Shin, K. and T. Kuboyama. 2008. A generalization of Haussler's convolution kernel: mapping kernel. In *Proceedings of the 25th international conference on Machine learning*, pages 944–951. ACM.

[Stolcke2002] Stolcke, A. 2002. Srilm-an extensible language modeling toolkit. In *Proceedings of the international conference on spoken language processing*, volume 2, pages 901–904. Citeseer.

[Sun, Zhang, and Tan2009] Sun, J., M. Zhang, and C.L. Tan. 2009. A non-contiguous tree sequence alignment-based model for statistical machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 914–922. Association for Computational Linguistics.

[Sun, Zhang, and Tan2010a] Sun, J., M. Zhang, and C.L. Tan. 2010a. Discriminative induction of sub-tree alignment using limited labeled data. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1047–1055. Association for Computational Linguistics.

[Sun, Zhang, and Tan2010b] Sun, J., M. Zhang, and C.L. Tan. 2010b. Exploring syntactic structural features for sub-tree alignment using bilingual tree kernels. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 306–315. Association for Computational Linguistics.

[Sun, Zhang, and Tan2011] Sun, J., M. Zhang, and C.L. Tan. 2011. Tree sequence kernel for natural language. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*.

[Suzuki, Sasaki, and Maeda2006] Suzuki, J., Y. Sasaki, and E. Maeda. 2006. Hierarchical directed acyclic graph kernel. *Systems and Computers in Japan*, 37(10):58–68.

[Suzuki et al.2003] Suzuki, J., H. Taira, Y. Sasaki, and E. Maeda. 2003. Question

classification using hdag kernel. In *Proceedings of the ACL 2003 workshop on Multilingual summarization and question answering-Volume 12*, pages 61–68. Association for Computational Linguistics.

[Tinsley, Hearne, and Way2009] Tinsley, J., M. Hearne, and A. Way. 2009. Parallel treebanks in phrase-based statistical machine translation. *Proceedings of CICLING-09*.

[Tinsley et al.2007] Tinsley, J., V. Zhechev, M. Hearne, and A. Way. 2007. Robust language-pair independent sub-tree alignment. *Machine Translation Summit XI*, pages 467–474.

[Vapnik1998] Vapnik, V.N. 1998. Statistical learning theory.

[Watanabe, Kurohashi, and Aramaki2000] Watanabe, H., S. Kurohashi, and E. Aramaki. 2000. Finding structural correspondences from bilingual parsed corpus for corpus-based translation. In *Proceedings of the 18th conference on Computational linguistics-Volume 2*, pages 906–912. Association for Computational Linguistics.

[Yamamoto and Matsumoto2000] Yamamoto, K. and Y. Matsumoto. 2000. Acquisition of phrase-level bilingual correspondence using dependency structure. In *Proceedings of the 18th conference on Computational linguistics-Volume 2*, pages 933–939. Association for Computational Linguistics.

[Yang, Su, and Tan2006] Yang, X., J. Su, and C.L. Tan. 2006. Kernel-based pronoun resolution with structured syntactic knowledge. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 41–48. Association for Computational Linguistics.

[Zanzotto, Pennacchiotti, and Moschitti2009] Zanzotto, F., M. Pennacchiotti, and A. Moschitti. 2009. A machine learning approach to textual entailment recognition. *Natural Language Engineering*, 15(04):551–582.

[Zanzotto and Moschitti2006] Zanzotto, F.M. and A. Moschitti. 2006. Automatic learning of textual entailments with cross-pair similarities. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 401–408. Association for Computational Linguistics.

[Zelenko, Aone, and Richardella2003] Zelenko, D., C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *The Journal of Machine Learning Research*, 3:1083–1106.

[Zhang and Lee2003] Zhang, D. and W.S. Lee. 2003. Question classification using support vector machines. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 26–32. ACM.

[Zhang et al.2008a] Zhang, M., W. Che, G.D. Zhou, A. Aw, C.L. Tan, T. Liu, and S. Li. 2008a. Semantic role labeling using a grammar-driven convolution tree kernel. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(7):1315–1329.

[Zhang et al.2008b] Zhang, M., H. Jiang, A. Aw, H. Li, C.L. Tan, and S. Li. 2008b. A tree sequence alignment-based tree-to-tree translation model. *Proc. ACL-08: HLT*, pages 559–567.

[Zhang et al.2007] Zhang, M., H. Jiang, A.T. Aw, J. Sun, S. Li, and C.L. Tan. 2007. A tree-to-tree alignment-based model for statistical machine translation. *MT-Summit-07*, pages 535–542.

[Zhang, Zhang, and Li2010] Zhang, M., H. Zhang, and H. Li. 2010. Convolution kernel over packed parse forest. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 875–885. Association for Computational Linguistics.

[Zhang, Zhou, and Aw2008] Zhang, M., G.D. Zhou, and A. Aw. 2008. Exploring

syntactic structured features over parse trees for relation extraction using kernel methods. *Information Processing & Management*, 44(2):687–701.

[Zhao and Grishman2005] Zhao, S. and R. Grishman. 2005. Extracting relations with integrated information using kernel methods. In *Proceedings of the 43rd Annual Meeting of Association for Computational Linguistics*, pages 419–426. Association for Computational Linguistics.

[Zhou and Zhang2007] Zhou, G. and M. Zhang. 2007. Extracting relation information from text documents by exploring various types of knowledge. *Information Processing & Management*, 43(4):969–982.

[Zhou et al.2007] Zhou, G., M. Zhang, D. Hong, and J.Q. Zhu. 2007. Tree kernel-based relation extraction with context-sensitive structured parse tree information. In *Proceedings of the 2007 conference on Empirical methods in natural language processing*. Association for Computational Linguistics.