String Re-writing Kernel

Fan Bu[1], **Hang Li**[2] and **Xiaoyan Zhu**[3]
[1,3]State Key Laboratory of Intelligent Technology and Systems
[1,3]Tsinghua National Laboratory for Information Sci. and Tech.
[1,3]Department of Computer Sci. and Tech., Tsinghua University
[2]Microsoft Research Asia, No. 5 Danling Street, Beijing 100080,China
[1]bufan0000@gmail.com
[2]hangli@microsoft.com
[3]zxy-dcs@tsinghua.edu.cn

Abstract

Learning for sentence re-writing is a fundamental task in natural language processing and information retrieval. In this paper, we propose a new class of kernel functions, referred to as string re-writing kernel, to address the problem. A string re-writing kernel measures the similarity between *two pairs of strings*, each pair representing re-writing of a string. It can capture the lexical and structural similarity between two pairs of sentences without the need of constructing syntactic trees. We further propose an instance of string re-writing kernel which can be computed efficiently. Experimental results on benchmark datasets show that our method can achieve better results than state-of-the-art methods on two sentence re-writing learning tasks: paraphrase identification and recognizing textual entailment.

1 Introduction

Learning for sentence re-writing is a fundamental task in natural language processing and information retrieval, which includes paraphrasing, textual entailment and transformation between query and document title in search.

The key question here is how to represent the re-writing of sentences. In previous research on sentence re-writing learning such as paraphrase identification and recognizing textual entailment, most representations are based on the lexicons (Zhang and Patrick, 2005; Lintean and Rus, 2011; de Marneffe et al., 2006) or the syntactic trees (Das and Smith, 2009; Heilman and Smith, 2010) of the sentence pairs.

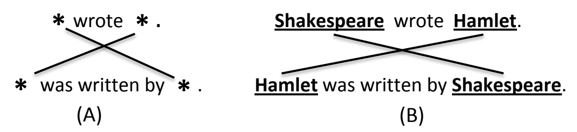

Figure 1: Example of re-writing. (A) is a re-writing rule and (B) is a re-writing of sentence.

In (Lin and Pantel, 2001; Barzilay and Lee, 2003), re-writing rules serve as underlying representations for paraphrase generation/discovery. Motivated by the work, we represent re-writing of sentences by all possible re-writing rules that can be applied into it. For example, in Fig. 1, (A) is one re-writing rule that can be applied into the sentence re-writing (B). Specifically, we propose a new class of kernel functions (Schölkopf and Smola, 2002), called string re-writing kernel (SRK), which defines the similarity between two re-writings (pairs) of strings as the inner product between them in the feature space induced by all the re-writing rules. SRK is different from existing kernels in that it is for re-writing and defined on *two pairs of strings*. SRK can capture the lexical and structural similarity between re-writings of sentences and does not need to parse the sentences and create the syntactic trees of them.

One challenge for using SRK lies in the high computational cost of straightforwardly computing the kernel, because it involves two re-writings of strings (i.e., four strings) and a large number of re-writing rules. We are able to develop an instance of SRK, referred to as kb-SRK, which directly computes the number of common rewriting rules without explicitly calculating the inner product between feature vectors, and thus drastically reduce the time complexity.

Experimental results on benchmark datasets show that SRK achieves better results than the state-of-the-art methods in paraphrase identification and recognizing textual entailment. Note that SRK is very flexible to the formulations of sentences. For example, informally written sentences such as long queries in search can also be effectively handled.

2 Related Work

The string kernel function, first proposed by Lodhi et al. (2002), measures the similarity between two strings by their shared substrings. Leslie et al. (2002) proposed the k-spectrum kernel which represents strings by their contiguous substrings of length k. Leslie et al. (2004) further proposed a number of string kernels including the wildcard kernel to facilitate inexact matching between the strings. The string kernels defined on two pairs of objects (including strings) were also developed, which decompose the similarity into product of similarities between individual objects using tensor product (Basilico and Hofmann, 2004; Ben-Hur and Noble, 2005) or Cartesian product (Kashima et al., 2009).

The task of paraphrasing usually consists of paraphrase pattern generation and paraphrase identification. Paraphrase pattern generation is to automatically extract semantically equivalent patterns (Lin and Pantel, 2001; Bhagat and Ravichandran, 2008) or sentences (Barzilay and Lee, 2003). Paraphrase identification is to identify whether two given sentences are a paraphrase of each other. The methods proposed so far formalized the problem as classification and used various types of features such as bag-of-words feature, edit distance (Zhang and Patrick, 2005), dissimilarity kernel (Lintean and Rus, 2011) predicate-argument structure (Qiu et al., 2006), and tree edit model (which is based on a tree kernel) (Heilman and Smith, 2010) in the classification task. Among the most successful methods, Wan et al. (2006) enriched the feature set by the BLEU metric and dependency relations. Das and Smith (2009) used the quasi-synchronous grammar formalism to incorporate features from WordNet, named entity recognizer, POS tagger, and dependency labels from aligned trees.

The task of recognizing textual entailment is to decide whether the hypothesis sentence can be entailed by the premise sentence (Giampiccolo et al., 2007). In recognizing textual entailment, de Marneffe et al. (2006) classified sentences pairs on the basis of word alignments. MacCartney and Manning (2008) used an inference procedure based on natural logic and combined it with the methods by de Marneffe et al. (2006). Harmeling (2007) and Heilman and Smith (2010) classified sequence pairs based on transformation on syntactic trees. Zanzotto et al. (2007) used a kernel method on syntactic tree pairs (Moschitti and Zanzotto, 2007).

3 Kernel Approach to Sentence Re-Writing Learning

We formalize sentence re-writing learning as a kernel method. Following the literature of string kernel, we use the terms "string" and "character" instead of "sentence" and "word".

Suppose that we are given training data consisting of re-writings of strings and their responses

$$((s_1,t_1),y_1),...,((s_n,t_n),y_n) \in (\Sigma^* \times \Sigma^*) \times \mathrm{Y}$$

where Σ denotes the character set, $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$ denotes the string set, which is the Kleene closure of set Σ, Y denotes the set of responses, and n is the number of instances. (s_i,t_i) is a re-writing consisting of the source string s_i and the target string t_i. y_i is the response which can be a category, ordinal number, or real number. In this paper, for simplicity we assume that $\mathrm{Y} = \{\pm 1\}$ (e.g. paraphrase/non-paraphrase). Given a new string re-writing $(s,t) \in \Sigma^* \times \Sigma^*$, our goal is to predict its response y. That is, the training data consists of binary classes of string re-writings, and the prediction is made for the new re-writing based on learning from the training data.

We take the kernel approach to address the learning task. The kernel on re-writings of strings is defined as

$$\mathrm{K} : (\Sigma^* \times \Sigma^*) \times (\Sigma^* \times \Sigma^*) \to \mathbb{R}$$

satisfying for all (s_i,t_i), $(s_j,t_j) \in \Sigma^* \times \Sigma^*$,

$$\mathrm{K}((s_i,t_i),(s_j,t_j)) = \langle \Phi(s_i,t_i), \Phi(s_j,t_j) \rangle$$

where Φ maps each re-writing (pair) of strings into a high dimensional Hilbert space $\mathscr{H}$, referred to as feature space. By the representer theorem (Kimeldorf and Wahba, 1971; Schölkopf and Smola, 2002), it can be shown that the response y of a new string re-writing (s,t) can always be represented as

$$y = \mathrm{sign}(\sum_{i=1}^{n} \alpha_i y_i \mathrm{K}((s_i,t_i),(s,t)))$$

where $\alpha_i \geq 0, (i = 1, \cdots, n)$ are parameters. That is, it is determined by a linear combination of the similarities between the new instance and the instances in training set. It is also known that by employing a learning model such as SVM (Vapnik, 2000), such a linear combination can be automatically learned by solving a quadratic optimization problem. The question then becomes how to design the kernel function for the task.

4 String Re-writing Kernel

Let Σ be the set of characters and Σ^* be the set of strings. Let wildcard domain $\mathrm{D} \subseteq \Sigma^*$ be the set of strings which can be replaced by wildcards.

The string re-writing kernel measures the similarity between two string re-writings through the re-writing rules that can be applied into them. Formally, given re-writing rule set R and wildcard domain D, the *string re-writing kernel (SRK)* is defined as

$$\mathrm{K}((s_1,t_1),(s_2,t_2)) = \langle \Phi(s_1,t_1), \Phi(s_2,t_2) \rangle \qquad (1)$$

where $\Phi(s,t) = (\phi_r(s,t))_{r \in \mathrm{R}}$ and

$$\phi_r(s,t) = n\lambda^i \qquad (2)$$

where n is the number of contiguous substring pairs of (s,t) that re-writing rule r matches, i is the number of wildcards in r, and $\lambda \in (0,1]$ is a factor punishing each occurrence of wildcard.

A *re-writing rule* is defined as a triple $r = (\beta_s, \beta_t, \tau)$ where $\beta_s, \beta_t \in (\Sigma \cup \{*\})^*$ denote source and target string patterns and $\tau \subseteq ind_*(\beta_s) \times ind_*(\beta_t)$ denotes the alignments between the wildcards in the two string patterns. Here $ind_*(\beta)$ denotes the set of indexes of wildcards in β.

We say that a re-writing rule (β_s, β_t, τ) *matches* a string pair (s,t), if and only if string patterns β_s and β_t can be changed into s and t respectively by substituting each wildcard in the string patterns with an element in the strings, where the elements are defined in the wildcard domain D and the wildcards $\beta_s[i]$ and $\beta_t[j]$ are substituted by the same elements, when there is an alignment $(i,j) \in \tau$.

For example, the re-writing rule in Fig. 1 (A) can be formally written as $r = (\beta s, \beta t, \tau)$ where $\beta s = (*, wrote, *)$, $\beta t = (*, was, written, by, *)$ and $\tau = \{(1,5),(3,1)\}$. It matches with the string pair in Fig. 1 (B).

String re-writing kernel is a class of kernels which depends on re-writing rule set R and wildcard domain D. Here we provide some examples. Obviously, the effectiveness and efficiency of SRK depend on the choice of R and D.

Example 1. *We define the pairwise k-spectrum kernel (ps-SRK) K_k^{ps} as the re-writing rule kernel under $R = \{(\beta_s, \beta_t, \tau) | \beta_s, \beta_t \in \Sigma^k, \tau = \emptyset\}$ and any D. It can be shown that $K_k^{ps}((s_1,t_1),(s_2,t_2)) = K_k^{spec}(s_1,s_2)K_k^{spec}(t_1,t_2)$ where $K_k^{spec}(x,y)$ is equivalent to the k-spectrum kernel proposed by Leslie et al. (2002).*

Example 2. *The pairwise k-wildcard kernel (pw-SRK) K_k^{pw} is defined as the re-writing rule kernel under $R = \{(\beta_s, \beta_t, \tau) | \beta_s, \beta_t \in (\Sigma \cup \{*\})^k, \tau = \emptyset\}$ and $D = \Sigma$. It can be shown that $K_k^{pw}((s_1,t_1),(s_2,t_2)) = K_{(k,k)}^{wc}(s_1,s_2)K_{(k,k)}^{wc}(t_1,t_2)$ where $K_{(k,k)}^{wc}(x,y)$ is a special case (m=k) of the (k,m)-wildcard kernel proposed by Leslie et al. (2004).*

Both kernels shown above are represented as the product of two kernels defined separately on strings s_1, s_2 and t_1, t_2, and that is to say that they do not consider the alignment relations between the strings.

5 K-gram Bijective String Re-writing Kernel

Next we propose another instance of string re-writing kernel, called the k-gram bijective string re-writing kernel (kb-SRK). As will be seen, kb-SRK can be computed efficiently, although it is defined on two pairs of strings and is not decomposed (note that ps-SRK and pw-SRK are decomposed).

5.1 Definition

The kb-SRK has the following properties: (1) A wildcard can only substitute a single character, denoted as "?". (2) The two string patterns in a re-writing rule are of length k. (3) The alignment relation in a re-writing rule is bijective, i.e., there is a one-to-one mapping between the wildcards in the string patterns. Formally, the *k-gram bijective string re-writing kernel* K_k is defined as a string re-writing kernel under the re-writing rule set $\mathrm{R} = \{(\beta_s, \beta_t, \tau) | \beta_s, \beta_t \in (\Sigma \cup \{?\})^k, \tau \text{ is bijective}\}$ and the wildcard domain $\mathrm{D} = \Sigma$.

Since each re-writing rule contains two string patterns of length k and each wildcard can only substitute one character, a re-writing rule can only match k-gram pairs in (s,t). We can rewrite Eq. (2) as

$$\phi_r(s,t) = \sum_{\alpha_s \in \text{k-grams}(s)} \sum_{\alpha_t \in \text{k-grams}(t)} \bar{\phi}_r(\alpha_s, \alpha_t) \quad (3)$$

where $\bar{\phi}_r(\alpha_s, \alpha_t) = \lambda^i$ if r (with i wildcards) matches (α_s, α_t), otherwise $\bar{\phi}_r(\alpha_s, \alpha_t) = 0$.

For ease of computation, we re-write kb-SRK as

$$\mathrm{K}_k((s_1,t_1),(s_2,t_2)) = \sum_{\substack{\alpha_{s_1} \in \text{k-grams}(s_1) \\ \alpha_{t_1} \in \text{k-grams}(t_1)}} \sum_{\substack{\alpha_{s_2} \in \text{k-grams}(s_2) \\ \alpha_{t_2} \in \text{k-grams}(t_2)}} \bar{\mathrm{K}}_k((\alpha_{s_1}, \alpha_{t_1}), (\alpha_{s_2}, \alpha_{t_2})) \quad (4)$$

where

$$\bar{\mathrm{K}}_k = \sum_{r \in R} \bar{\phi}_r(\alpha_{s_1}, \alpha_{t_1}) \bar{\phi}_r(\alpha_{s_2}, \alpha_{t_2}) \quad (5)$$

5.2 Algorithm for Computing Kernel

A straightforward computation of kb-SRK would be intractable. The computation of K_k in Eq. (4) needs computations of $\bar{\mathrm{K}}_k$ conducted $O((n-k+1)^4)$ times, where n denotes the maximum length of strings. Furthermore, the computation of $\bar{\mathrm{K}}_k$ in Eq. (5) needs to perform matching of all the re-writing rules with the two k-gram pairs (α_{s_1}, α_{t_1}), (α_{s_2}, α_{t_2}), which has time complexity $O(k!)$.

In this section, we will introduce an efficient algorithm, which can compute $\bar{\mathrm{K}}_k$ and K_k with the time complexities of $O(k)$ and $O(kn^2)$, respectively. The latter is verified empirically.

5.2.1 Transformation of Problem

For ease of manipulation, our method transforms the computation of kernel on k-grams into the computation on a new data structure called lists of doubles. We first explain how to make the transformation.

Suppose that $\alpha_1, \alpha_2 \in \Sigma^k$ are k-grams, we use $\alpha_1[i]$ and $\alpha_2[i]$ to represent the i-th characters of them. We call a pair of characters a double. Thus $\Sigma \times \Sigma$ denotes the set of doubles and $\alpha_s^{\mathrm{D}}, \alpha_t^{\mathrm{D}} \in (\Sigma \times \Sigma)^k$ denote lists of doubles. The following operation combines two k-grams into a list of doubles.

$$\alpha_{s_1} = \text{abbccbb}\,; \qquad \alpha_{s_2} = \text{abcccdd};$$
$$\alpha_{t_1} = \text{cbcbbcb}\,; \qquad \alpha_{t_2} = \text{cbccdcd};$$

Figure 2: Example of two k-gram pairs.

$$\alpha_s^{\mathrm{D}} = (\mathrm{a},\mathrm{a}),(\mathrm{b},\mathrm{b}),(\mathbf{b},\mathbf{c}),(\mathrm{c},\mathrm{c}),(\mathrm{c},\mathrm{c}),(\mathbf{b},\mathbf{d}),(\mathbf{b},\mathbf{d})$$
$$\alpha_t^{\mathrm{D}} = (\mathrm{c},\mathrm{c}),(\mathrm{b},\mathrm{b}),(\mathrm{c},\mathrm{c}),(\mathbf{b},\mathbf{c}),(\mathbf{b},\mathbf{d}),(\mathrm{c},\mathrm{c}),(\mathbf{b},\mathbf{d})$$

Figure 3: Example of the pair of double lists combined from the two k-gram pairs in Fig. 2. Non-identical doubles are in bold.

$$\alpha_1 \otimes \alpha_2 = ((\alpha_1[1], \alpha_2[1]), \cdots, (\alpha_1[k], \alpha_2[k])).$$

We denotes $\alpha_1 \otimes \alpha_2[i]$ as the i-th element of the list. Fig. 3 shows example lists of doubles combined from k-grams.

We introduce the set of *identical doubles* $\mathrm{I} = \{(c,c) | c \in \Sigma\}$ and the set of *non-identical doubles* $\mathrm{N} = \{(c,c') | c, c' \in \Sigma \text{ and } c \neq c'\}$. Obviously, $\mathrm{I} \bigcup \mathrm{N} = \Sigma \times \Sigma$ and $\mathrm{I} \bigcap \mathrm{N} = \emptyset$.

We define the set of *re-writing rules for double lists* $\mathrm{R}^{\mathrm{D}} = \{r^{\mathrm{D}} = (\beta_s^{\mathrm{D}}, \beta_t^{\mathrm{D}}, \tau) | \beta_s^{\mathrm{D}}, \beta_t^{\mathrm{D}} \in (\mathrm{I} \cup \{?\})^k, \tau$ is a bijective alignment$\}$ where β_s^{D} and β_t^{D} are lists of identical doubles including wildcards and with length k. We say rule r^{D} matches a pair of double lists $(\alpha_s^{\mathrm{D}}, \alpha_t^{\mathrm{D}})$ iff. $\beta_s^{\mathrm{D}}, \beta_t^{\mathrm{D}}$ can be changed into α_s^{D} and α_t^{D} by substituting each wildcard pair to a double in $\Sigma \times \Sigma$, and the double substituting the wildcard pair $\beta_s^{\mathrm{D}}[i]$ and $\beta_t^{\mathrm{D}}[j]$ must be an identical double when there is an alignment $(i,j) \in \tau$. The rule set defined here and the rule set in Sec. 4 only differ on the elements where re-writing occurs. Fig. 4 (B) shows an example of re-writing rule for double lists. The pair of double lists in Fig. 3 can match with the re-writing rule.

5.2.2 Computing $\bar{\mathrm{K}}_k$

We consider how to compute $\bar{\mathrm{K}}_k$ by extending the computation from k-grams to double lists.

The following lemma shows that computing the weighted sum of re-writing rules matching k-gram pairs $(\alpha_{s_1}, \alpha_{t_1})$ and $(\alpha_{s_2}, \alpha_{t_2})$ is equivalent to computing the weighted sum of re-writing rules for double lists matching $(\alpha_{s_1} \otimes \alpha_{s_2}, \alpha_{t_1} \otimes \alpha_{t_2})$.

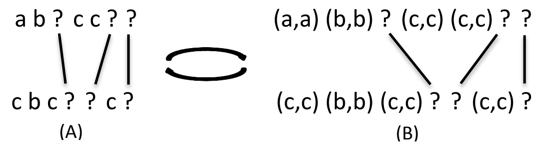

Figure 4: For re-writing rule (A) matching both k-gram pairs shown in Fig. 2, there is a corresponding re-writing rule for double lists (B) matching the pair of double lists shown in Fig. 3.

$\#_{\Sigma\times\Sigma}(\alpha_s^D) = \{(a,a){:}\,1, (b,b){:}\,1, (\mathbf{b},\mathbf{c}){:}\,1, (\mathbf{b},\mathbf{d}){:}\,2, (c,c){:}\,2\}$

$\#_{\Sigma\times\Sigma}(\alpha_t^D) = \{(a,a){:}\,0, (b,b){:}\,1, (\mathbf{b},\mathbf{c}){:}\,1, (\mathbf{b},\mathbf{d}){:}\,2, (c,c){:}\,3\}$

Figure 5: Example of $\#_{\Sigma\times\Sigma}(\cdot)$ for the two double lists shown in Fig. 3. Doubles not appearing in both α_s^{D} and α_t^{D} are not shown.

Algorithm 1: Computing $\bar{\mathrm{K}}_k$

```
Input: k-gram pair $(\alpha_{s_1}, \alpha_{t_1})$ and $(\alpha_{s_2}, \alpha_{t_2})$
Output: $\bar{\mathrm{K}}_k((\alpha_{s_1}, \alpha_{t_1}), (\alpha_{s_2}, \alpha_{t_2}))$
1 Set $(\alpha_s^{\mathrm{D}}, \alpha_t^{\mathrm{D}}) = (\alpha_{s_1} \otimes \alpha_{s_2}, \alpha_{t_1} \otimes \alpha_{t_2})$ ;
2 Compute $\#_{\Sigma\times\Sigma}(\alpha_s^{\mathrm{D}})$ and $\#_{\Sigma\times\Sigma}(\alpha_t^{\mathrm{D}})$;
3 result=1;
4 for each $e \in \Sigma \times \Sigma$ satisfies
    $\#_e(\alpha_s^{\mathrm{D}}) + \#_e(\alpha_t^{\mathrm{D}}) \neq 0$ do
5     $g_e = 0, n_e = \min\{\#_e(\alpha_s^{\mathrm{D}}), \#_e(\alpha_t^{\mathrm{D}})\}$ ;
6     for $0 \le i \le n_e$ do
7         $g_e = g_e + a_i^{(e)} \lambda^{2i}$;
8     $result = result * g$;
9 return result;
```

Lemma 1. *For any two k-gram pairs $(\alpha_{s_1}, \alpha_{t_1})$ and $(\alpha_{s_2}, \alpha_{t_2})$, there exists a one-to-one mapping from the set of re-writing rules matching them to the set of re-writing rules matching the corresponding double lists $(\alpha_{s_1} \otimes \alpha_{s_2}, \alpha_{t_1} \otimes \alpha_{t_2})$.*

The re-writing rule in Fig. 4 (A) matches the k-gram pairs in Fig. 2. Equivalently, the re-writing rule for double lists in Fig. 4 (B) matches the pair of double lists in Fig. 3. By lemma 1 and Eq. 5, we have

$$\bar{\mathrm{K}}_k = \sum_{r^D \in R^D} \bar{\phi}_{r^D}(\alpha_{s_1} \otimes \alpha_{s_2}, \alpha_{t_1} \otimes \alpha_{t_2}) \qquad (6)$$

where $\bar{\phi}_{r^D}(\alpha_s^D, \alpha_t^D) = \lambda^{2i}$ if the rewriting rule for double lists r^D with i wildcards matches (α_s^D, α_t^D), otherwise $\bar{\phi}_{r^D}(\alpha_s^D, \alpha_t^D) = 0$. To get $\bar{\mathrm{K}}_k$, we just need to compute the weighted sum of re-writing rules for double lists matching $(\alpha_{s_1} \otimes \alpha_{s_2}, \alpha_{t_1} \otimes \alpha_{t_2})$. Thus, we can work on the "combined" pair of double lists instead of two pairs of *k*-grams.

Instead of enumerating all possible re-writing rules and checking whether they can match the given pair of double lists, we only calculate the number of possibilities of "generating" from the pair of double lists to the re-writing rules matching it, which can be carried out efficiently. We say that a re-writing rule of double lists can be generated from a pair of double lists (α_s^{D}, α_t^{D}), if they match with each other. From the definition of R^{D}, in each generation, the identical doubles in α_s^{D} and α_t^{D} can be either or not substituted by an aligned wildcard pair in the re-writing rule, and all the non-identical doubles in α_s^{D} and α_t^{D} must be substituted by aligned wildcard pairs. From this observation and Eq. 6, $\bar{\mathrm{K}}_k$ only depends on the number of times each double occurs in the double lists.

Let e be a double. We denote $\#_e(\alpha^{\mathrm{D}})$ as the number of times e occurs in the list of doubles α^{D}. Also, for a set of doubles $\mathrm{S} \subseteq \Sigma \times \Sigma$, we denote $\#_{\mathrm{S}}(\alpha^{\mathrm{D}})$ as a vector in which each element represents $\#_e(\alpha^{\mathrm{D}})$ of each double $e \in \mathrm{S}$. We can find a function g such that

$$\bar{\mathrm{K}}_k = g(\#_{\Sigma\times\Sigma}(\alpha_{s_1} \otimes \alpha_{s_2}), \#_{\Sigma\times\Sigma}(\alpha_{t_1} \otimes \alpha_{t_2})) \qquad (7)$$

Alg. 1 shows how to compute $\bar{K}_k$. $\#_{\Sigma\times\Sigma}(.)$ is computed from the two pairs of k-grams in line 1-2. The final score is made through the iterative calculation on the two lists (lines 4-8).

The key of Alg. 1 is the calculation of g_e based on $a_i^{(e)}$ (line 7). Here we use $a_i^{(e)}$ to denote the number of possibilities for which i pairs of aligned wildcards can be generated from e in both α_s^{D} and α_t^{D}. $a_i^{(e)}$ can be computed as follows.

(1) If $e \in \mathrm{N}$ and $\#_e(\alpha_s^{\mathrm{D}}) \neq \#_e(\alpha_t^{\mathrm{D}})$, then $a_i^{(e)} = 0$ for any i.

(2) If $e \in \mathrm{N}$ and $\#_e(\alpha_s^{\mathrm{D}}) = \#_e(\alpha_t^{\mathrm{D}}) = j$, then $a_j^{(e)} = j!$ and $a_i^{(e)} = 0$ for any $i \neq j$.

(3) If $e \in \mathrm{I}$, then $a_i^{(e)} = \binom{\#_e(\alpha_s^{\mathrm{D}})}{i}\binom{\#_e(\alpha_t^{\mathrm{D}})}{i} i!$.

We next explain the rationale behind the above computations. In (1), since $\#_e(\alpha_s^{\mathrm{D}}) \neq \#_e(\alpha_t^{\mathrm{D}})$, it is impossible to generate a re-writing rule in which all the occurrences of non-identical double e are substituted by pairs of aligned wildcards. In (2), j pairs of aligned wildcards can be generated from all the occurrences of non-identical double e in both α_s^{D} and α_t^{D}. The number of combinations thus is $j!$. In (3), a pair of aligned wildcards can either be generated or not from a pair of identical doubles in α_s^{D} and α_t^{D}. We can select i occurrences of identical double e from α_s^{D}, i occurrences from α_t^{D}, and generate all possible aligned wildcards from them.

In the loop of lines 4-8, we only need to consider $a_i^{(e)}$ for $0 \le i \le \min\{\#_e(\alpha_s^{\mathrm{D}}), \#_e(\alpha_t^{\mathrm{D}})\}$, because $a_i^{(e)} = 0$ for the rest of i.

To sum up, Eq. 7 can be computed as below, which is exactly the computation at lines 3-8.

$$g(\#_{\Sigma\times\Sigma}(\alpha_s^{\mathrm{D}}), \#_{\Sigma\times\Sigma}(\alpha_t^{\mathrm{D}})) = \prod_{e\in\Sigma\times\Sigma} (\sum_{i=0}^{n_e} a_i^{(e)} \lambda^{2i}) \quad (8)$$

For the k-gram pairs in Fig. 2, we first create lists of doubles in Fig. 3 and compute $\#_{\Sigma\times\Sigma}(\cdot)$ for them (lines 1-2 of Alg. 1), as shown in Fig. 5. We next compute K_k from $\#_{\Sigma\times\Sigma}(\alpha_s^{\mathrm{D}})$ and $\#_{\Sigma\times\Sigma}(\alpha_t^{\mathrm{D}})$ in Fig. 5 (lines 3-8 of Alg. 1) and obtain $\mathrm{K}_k = (1)(1+\lambda^2)(\lambda^2)(2\lambda^4)(1+6\lambda^2+6\lambda^4) = 12\lambda^{12}+24\lambda^{10}+14\lambda^8+2\lambda^6$.

5.2.3 Computing K_k

Algorithm 2 shows how to compute K_k. It prepares two maps m_s and m_t and two vectors of counters c_s and c_t. In m_s and m_t, each key $\#_{\mathrm{N}}(.)$ maps a set of values $\#_{\Sigma\times\Sigma}(.)$. Counters c_s and c_t count the frequency of each $\#_{\Sigma\times\Sigma}(.)$. Recall that $\#_{\mathrm{N}}(\alpha_{s_1}\otimes\alpha_{s_2})$ denotes a vector whose element is $\#_e(\alpha_{s_1}\otimes\alpha_{s_2})$ for $e \in \mathrm{N}$. $\#_{\Sigma\times\Sigma}(\alpha_{s_1}\otimes\alpha_{s_2})$ denotes a vector whose element is $\#_e(\alpha_{s_1}\otimes\alpha_{s_2})$ where e is any possible double.

One can easily verify the output of the algorithm is exactly the value of K_k. First, $\bar{\mathrm{K}}_k((\alpha_{s_1},\alpha_{t_1}),(\alpha_{s_2},\alpha_{t_2})) = 0$ if $\#_{\mathrm{N}}(\alpha_{s_1}\otimes\alpha_{s_2}) \neq \#_{\mathrm{N}}(\alpha_{t_1}\otimes\alpha_{t_2})$. Therefore, we only need to consider those $\alpha_{s_1}\otimes\alpha_{s_2}$ and $\alpha_{t_1}\otimes\alpha_{t_2}$ which have the same key (lines 10-13). We group the k-gram pairs by their key in lines 2-5 and lines 6-9.

Moreover, the following relation holds

$$\bar{\mathrm{K}}_k((\alpha_{s_1},\alpha_{t_1}),(\alpha_{s_2},\alpha_{t_2})) = \bar{\mathrm{K}}_k((\alpha'_{s_1},\alpha'_{t_1}),(\alpha'_{s_2},\alpha'_{t_2}))$$

if $\#_{\Sigma\times\Sigma}(\alpha_{s_1}\otimes\alpha_{s_2}) = \#_{\Sigma\times\Sigma}(\alpha'_{s_1}\otimes\alpha'_{s_2})$ and $\#_{\Sigma\times\Sigma}(\alpha_{t_1}\otimes\alpha_{t_2}) = \#_{\Sigma\times\Sigma}(\alpha'_{t_1}\otimes\alpha'_{t_2})$, where α'_{s_1}, α'_{s_2}, α'_{t_1}, α'_{t_2} are

Algorithm 2: Computing K_k

```
   Input: string pair $(s_1,t_1)$ and $(s_2,t_2)$, window
          size $k$
   Output: $\mathrm{K}_k((s_1,t_1),(s_2,t_2))$
1  Initialize two maps $\mathrm{m}_s$ and $\mathrm{m}_t$ and two counters
   $\mathrm{c}_s$ and $\mathrm{c}_t$;
2  for each k-gram $\alpha_{s_1}$ in $s_1$ do
3      for each k-gram $\alpha_{s_2}$ in $s_2$ do
4          Update $\mathrm{m}_s$ with key-value pair
           $(\#_{\mathrm{N}}(\alpha_{s_1}\otimes\alpha_{s_2}), \#_{\Sigma\times\Sigma}(\alpha_{s_1}\otimes\alpha_{s_2}))$;
5          $\mathrm{c}_s[\#_{\Sigma\times\Sigma}(\alpha_{s_1}\otimes\alpha_{s_2})]++$ ;
6  for each k-gram $\alpha_{t_1}$ in $t_1$ do
7      for each k-gram $\alpha_{t_2}$ in $t_2$ do
8          Update $\mathrm{m}_t$ with key-value pair
           $(\#_{\mathrm{N}}(\alpha_{t_1}\otimes\alpha_{t_2}), \#_{\Sigma\times\Sigma}(\alpha_{t_1}\otimes\alpha_{t_2}))$;
9          $\mathrm{c}_t[\#_{\Sigma\times\Sigma}(\alpha_{t_1}\otimes\alpha_{t_2})]++$ ;
10 for each key $\in \mathrm{m}_s$.keys $\cap\ \mathrm{m}_t$.keys do
11     for each $v_s \in \mathrm{m}_s[key]$ do
12         for each $v_t \in \mathrm{m}_t[key]$ do
13             result+= $\mathrm{c}_s[v_s]\mathrm{c}_t[v_t]g(v_s,v_t)$ ;
14 return result;
```

other k-grams. Therefore, we only need to take $\#_{\Sigma\times\Sigma}(\alpha_{s_1}\otimes\alpha_{s_2})$ and $\#_{\Sigma\times\Sigma}(\alpha_{t_1}\otimes\alpha_{t_2})$ as the value under each key and count its frequency. That is to say, $\#_{\Sigma\times\Sigma}$ provides sufficient statistics for computing $\bar{\mathrm{K}}_k$.

The quantity $g(v_s,v_t)$ in line 13 is computed by Alg. 1 (lines 3-8).

5.3 Time Complexity

The time complexities of Alg. 1 and Alg. 2 are shown below.

For Alg. 1, lines 1-2 can be executed in $O(k)$. The time for executing line 7 is less than $\#_e(\alpha_s^{\mathrm{D}}) + \#_e(\alpha_t^{\mathrm{D}}) + 1$ for each e satisfying $\#_e(\alpha_s^{\mathrm{D}}) \neq 0$ or $\#_e(\alpha_t^{\mathrm{D}}) \neq 0$. Since $\sum_{e\in\Sigma\times\Sigma}\#_e(\alpha_s^{\mathrm{D}}) = \sum_{e\in\Sigma\times\Sigma}\#_e(\alpha_t^{\mathrm{D}}) = k$, the time for executing lines 3-8 is less than $4k$, which results in the $O(k)$ time complexity of Alg. 1.

For Alg. 2, we denote $n = \max\{|s_1|,|s_2|,|t_1|,|t_2|\}$. It is easy to see that if the maps and counters in the algorithm are implemented by hash maps, the time complexities of lines 2-5 and lines 6-9 are $O(kn^2)$. However, analyzing the time complexity of lines 10-

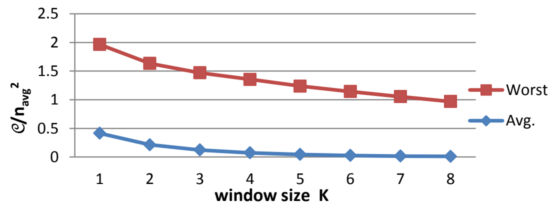

Figure 6: Relation between ratio $\mathfrak{C}/n_{avg}^2$ and window size k when running Alg. 2 on MSR Paraphrases Corpus.

13 is quite difficult.

Lemma 2 and Theorem 1 provide an upper bound of the number of times computing $g(v_s, v_t)$ in line 13, denoted as $\mathfrak{C}$.

Lemma 2. *For* $\alpha_{s_1} \in$*k-grams*(s_1) *and* $\alpha_{s_2}, \alpha'_{s_2} \in$*k-grams*$(s_2)$, *we have* $\#_{\Sigma\times\Sigma}(\alpha_{s_1} \otimes \alpha_{s_2}) = \#_{\Sigma\times\Sigma}(\alpha_{s_1} \otimes \alpha'_{s_2})$ *if* $\#_N(\alpha_{s_1} \otimes \alpha_{s_2}) = \#_N(\alpha_{s_1} \otimes \alpha'_{s_2})$.

Theorem 1. $\mathfrak{C}$ *is* $O(n^3)$.

By Lemma 2, each $\mathrm{m}_s[key]$ contains at most $n-k+1$ elements. Together with the fact that $\sum_{key} \mathrm{m}_s[key] = (n-k+1)^2$, Theorem 1 is proved. It can be also proved that $\mathfrak{C}$ is $O(n^2)$ when $k=1$.

Empirical study shows that $O(n^3)$ is a loose upper bound for $\mathfrak{C}$. Let n_{avg} denote the average length of s_1, t_1, s_2 and t_2. Our experiment on all pairs of sentences on MSR Paraphrase (Fig. 6) shows that $\mathfrak{C}$ is in the same order of n_{avg}^2 in the worst case and $\mathfrak{C}/n_{avg}^2$ decreases with increasing k in both average case and worst case, which indicates that $\mathfrak{C}$ is $O(n^2)$ and the overall time complexity of Alg. 2 is $O(kn^2)$.

6 Experiments

We evaluated the performances of the three types of string re-writing kernels on paraphrase identification and recognizing textual entailment: pairwise k-spectrum kernel (ps-SRK), pairwise k-wildcard kernel (pw-SRK), and k-gram bijective string re-writing kernel (kb-SRK). We set $\lambda = 1$ for all kernels. The performances were measured by accuracy (e.g. percentage of correct classifications).

In both experiments, we used LIBSVM with default parameters (Chang et al., 2011) as the classifier. All the sentences in the training and test sets were segmented into words by the tokenizer at OpenNLP (Baldrige et al.,). We further conducted stemming on the words with Iveonik English Stemmer (*http://www.iveonik.com/*).

We normalized each kernel by $\tilde{\mathrm{K}}(x,y) = \frac{\mathrm{K}(x,y)}{\sqrt{\mathrm{K}(x,x)\mathrm{K}(y,y)}}$ and then tried them under different window sizes k. We also tried to combine the kernels with two lexical features "unigram precision and recall" proposed in (Wan et al., 2006), referred to as PR. For each kernel K, we tested the window size settings of $\mathrm{K}_1 + ... + \mathrm{K}_{k_{max}}$ ($k_{max} \in \{1,2,3,4\}$) together with the combination with PR and we report the best accuracies of them in Tab 1 and Tab 2.

6.1 Paraphrase Identification

The task of paraphrase identification is to examine whether two sentences have the same meaning. We trained and tested all the methods on the MSR Paraphrase Corpus (Dolan and Brockett, 2005; Quirk et al., 2004) consisting of 4,076 sentence pairs for training and 1,725 sentence pairs for testing.

The experimental results on different SRKs are shown in Table 1. It can be seen that kb-SRK outperforms ps-SRK and pw-SRK. The results by the state-of-the-art methods reported in previous work are also included in Table 1. kb-SRK outperforms the existing lexical approach (Zhang and Patrick, 2005) and kernel approach (Lintean and Rus, 2011). It also works better than the other approaches listed in the table, which use syntactic trees or dependency relations.

Fig. 7 gives detailed results of the kernels under different maximum k-gram lengths k_{max} with and without PR. The results of ps-SRK and pw-SRK without combining PR under different k are all below 71%, therefore they are not shown for clar-

Method	**Acc.**
Zhang and Patrick (2005)	71.9
Lintean and Rus (2011)	73.6
Heilman and Smith (2010)	73.2
Qiu et al. (2006)	72.0
Wan et al. (2006)	75.6
Das and Smith (2009)	73.9
Das and Smith (2009)(PoE)	76.1
Our baseline (PR)	73.6
Our method (ps-SRK)	75.6
Our method (pw-SRK)	75.0
Our method (kb-SRK)	**76.3**

Table 1: Comparison with state-of-the-arts on MSRP.

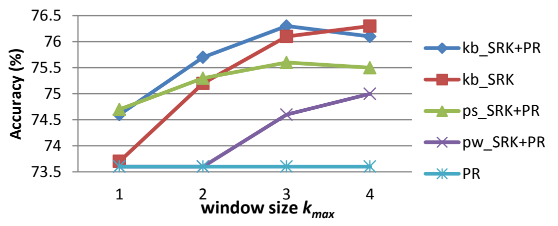

Figure 7: Performances of different kernels under different maximum window size k_{max} on MSRP.

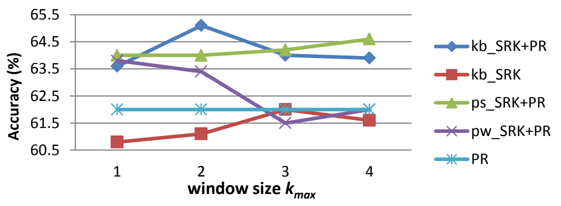

Figure 8: Performances of different kernels under different maximum window size k_{max} on RTE-3.

ity. By comparing the results of kb-SRK and pw-SRK we can see that the bijective property in kb-SRK is really helpful for improving the performance (note that both methods use wildcards). Furthermore, the performances of kb-SRK with and without combining PR increase dramatically with increasing k_{max} and reach the peaks (better than state-of-the-art) when k_{max} is four, which shows the power of the lexical and structural similarity captured by kb-SRK.

6.2 Recognizing Textual Entailment

Recognizing textual entailment is to determine whether a sentence (sometimes a short paragraph) can entail the other sentence (Giampiccolo et al., 2007). RTE-3 is a widely used benchmark dataset. Following the common practice, we combined the development set of RTE-3 and the whole datasets of RTE-1 and RTE-2 as training data and took the test set of RTE-3 as test data. The train and test sets contain 3,767 and 800 sentence pairs.

The results are shown in Table 2. Again, kb-SRK outperforms ps-SRK and pw-SRK. As indicated in (Heilman and Smith, 2010), the top-performing RTE systems are often built with significant engineering efforts. Therefore, we only compare with the six systems which involves less engineering. kb-SRK still outperforms most of those state-of-the-art methods even if it does not exploit any other lexical semantic sources and syntactic analysis tools.

Method	Acc.
Harmeling (2007)	59.5
de Marneffe et al. (2006)	60.5
M&M, (2007) (NL)	59.4
M&M, (2007) (Hybrid)	64.3
Zanzotto et al. (2007)	**65.75**
Heilman and Smith (2010)	62.8
Our baseline (PR)	62.0
Our method (ps-SRK)	64.6
Our method (pw-SRK)	63.8
Our method (kb-SRK)	65.1

Table 2: Comparison with state-of-the-arts on RTE-3.

Fig. 8 shows the results of the kernels under different parameter settings. Again, the results of ps-SRK and pw-SRK without combining PR are too low to be shown (all below 55%). We can see that PR is an effective method for this dataset and the overall performances are substantially improved after combining it with the kernels. The performance of kb-SRK reaches the peak when window size becomes two.

7 Conclusion

In this paper, we have proposed a novel class of kernel functions for sentence re-writing, called string re-writing kernel (SRK). SRK measures the lexical and structural similarity between two pairs of sentences without using syntactic trees. The approach is theoretically sound and is flexible to formulations of sentences. A specific instance of SRK, referred to as kb-SRK, has been developed which can balance the effectiveness and efficiency for sentence re-writing. Experimental results show that kb-SRK achieve better results than state-of-the-art methods on paraphrase identification and recognizing textual entailment.

Acknowledgments

This work is supported by the National Basic Research Program (973 Program) No. 2012CB316301.

References

Baldrige, J. , Morton, T. and Bierner G. *OpenNLP*. http://opennlp.sourceforge.net/.

Barzilay, R. and Lee, L. 2003. *Learning to paraphrase: An unsupervised approach using multiple-sequence alignment*. Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, pp. 16–23.

Basilico, J. and Hofmann, T. 2004. *Unifying collaborative and content-based filtering*. Proceedings of the twenty-first international conference on Machine learning, pp. 9, 2004.

Ben-Hur, A. and Noble, W.S. 2005. *Kernel methods for predicting protein–protein interactions*. Bioinformatics, vol. 21, pp. i38–i46, Oxford Univ Press.

Bhagat, R. and Ravichandran, D. 2008. *Large scale acquisition of paraphrases for learning surface patterns*. Proceedings of ACL-08: HLT, pp. 674–682.

Chang, C. and Lin, C. 2011. *LIBSVM: A library for support vector machines*. ACM Transactions on Intelligent Systems and Technology vol. 2, issue 3, pp. 27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

Das, D. and Smith, N.A. 2009. *Paraphrase identification as probabilistic quasi-synchronous recognition*. 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, pp. 468–476.

de Marneffe, M., MacCartney, B., Grenager, T., Cer, D., Rafferty A. and Manning C.D. 2006. *Learning to distinguish valid textual entailments*. Proc. of the Second PASCAL Challenges Workshop.

Dolan, W.B. and Brockett, C. 2005. *Automatically constructing a corpus of sentential paraphrases*. Proc. of IWP.

Giampiccolo, D., Magnini B., Dagan I., and Dolan B., editors 2007. *The third pascal recognizing textual entailment challenge*. Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing, pp. 1–9.

Harmeling, S. 2007. *An extensible probabilistic transformation-based approach to the third recognizing textual entailment challenge*. Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing, pp. 137–142, 2007.

Heilman, M. and Smith, N.A. 2010. *Tree edit models for recognizing textual entailments, paraphrases, and answers to questions*. Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 1011-1019.

Kashima, H. , Oyama, S. , Yamanishi, Y. and Tsuda, K. 2009. *On pairwise kernels: An efficient alternative and generalization analysis*. Advances in Knowledge Discovery and Data Mining, pp. 1030-1037, 2009, Springer.

Kimeldorf, G. and Wahba, G. 1971. *Some results on Tchebycheffian spline functions*. Journal of Mathematical Analysis and Applications, Vol.33, No.1, pp.82-95, Elsevier.

Lin, D. and Pantel, P. 2001. *DIRT-discovery of inference rules from text*. Proc. of ACM SIGKDD Conference on Knowledge Discovery and Data Mining.

Lintean, M. and Rus, V. 2011. *Dissimilarity Kernels for Paraphrase Identification*. Twenty-Fourth International FLAIRS Conference.

Leslie, C. , Eskin, E. and Noble, W.S. 2002. *The spectrum kernel: a string kernel for SVM protein classification*. Pacific symposium on biocomputing vol. 575, pp. 564-575, Hawaii, USA.

Leslie, C. and Kuang, R. 2004. *Fast string kernels using inexact matching for protein sequences*. The Journal of Machine Learning Research vol. 5, pp. 1435-1455.

Lodhi, H. , Saunders, C. , Shawe-Taylor, J. , Cristianini, N. and Watkins, C. 2002. *Text classification using string kernels*. The Journal of Machine Learning Research vol. 2, pp. 419-444.

MacCartney, B. and Manning, C.D. 2008. *Modeling semantic containment and exclusion in natural language inference*. Proceedings of the 22nd International Conference on Computational Linguistics, vol. 1, pp. 521-528, 2008.

Moschitti, A. and Zanzotto, F.M. 2007. *Fast and Effective Kernels for Relational Learning from Texts*. Proceedings of the 24th Annual International Conference on Machine Learning, Corvallis, OR, USA, 2007.

Qiu, L. and Kan, M.Y. and Chua, T.S. 2006. *Paraphrase recognition via dissimilarity significance classification*. Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, pp. 18–26.

Quirk, C. , Brockett, C. and Dolan, W. 2004. *Monolingual machine translation for paraphrase generation*. Proceedings of EMNLP 2004, pp. 142-149, Barcelona, Spain.

Schölkopf, B. and Smola, A.J. 2002. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. The MIT Press, Cambridge, MA.

Vapnik, V.N. 2000. *The nature of statistical learning theory*. Springer Verlag.

Wan, S. , Dras, M. , Dale, R. and Paris, C. 2006. *Using dependency-based features to take the "Para-farce" out of paraphrase*. Proc. of the Australasian Language Technology Workshop, pp. 131–138.

Zanzotto, F.M. , Pennacchiotti, M. and Moschitti, A. 2007. *Shallow semantics in fast textual entailment* *rule learners*. Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing, pp. 72–77.

Zhang, Y. and Patrick, J. 2005. *Paraphrase identification by text canonicalization*. Proceedings of the Australasian Language Technology Workshop, pp. 160–166.