An Introduction to 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]Huawei Noah's Ark Lab, Hong Kong
[1]bufan0000@gmail.com
[2]hangli.hl@huawei.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. It can capture the lexical and structural similarity between sentence pairs 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 comparable results with 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.

Motivated by previous work on paraphrase generation [Lin and Pantel, 2001; Barzilay and Lee, 2003], we represent a re-writing of sentence 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, 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 can capture the lexical and structural similarity between re-writings of sentences and does not need to parse the sentences to create the syntactic trees of them.

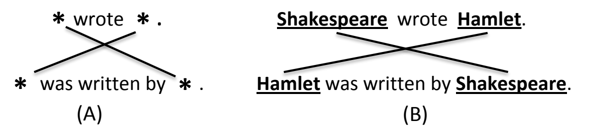

Figure 1: Example of re-writing. (A) is a re-writing rule and (B) is a re-writing of sentence.

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, for which we can efficiently count the number of applicable rewriting rules without explicitly enumerating all of them and thus can drastically reduce the time complexity of computation.

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

2 Related Work

A variety of kernels defined on two strings were proposed and applied to text classification and bioinformatics in previous work [Lodhi *et al.*, 2002; Leslie *et al.*, 2002; Leslie and Kuang, 2004]. For kernels defined on two pairs of objects (including strings), they were simply decomposed into the products of kernels between individual objects using tensor product [Basilico and Hofmann, 2004; Ben-Hur and Noble, 2005] or Cartesian product [Kashima *et al.*, 2009]. In this paper, we define the string re-wring kernel *on two pairs of strings*, but we do not necessarily decompose the kernel.

In paraphrase identification, 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] were used to conduct 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] utilized the quasi-synchronous grammar formalism to incorporate lexical and syntactic features. [Socher *et al.*, 2011] proposed deep learning approaches on paraphrase identification.

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] to achieve a comparative result. [Harmeling, 2007] and [Heilman and Smith, 2010] classified sequence pairs based on transformation on syntactic trees. [Zanzotto *et al.*, 2007] utilized a kernel method on syntactic tree pairs [Moschitti and Zanzotto, 2007] for the problem.

3 String Re-writing Kernel

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

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

Each training sample is a rewriting of strings and their response

$$((s,t),y) \in (\Sigma^* \times \Sigma^*) \times \mathrm{Y}.$$

In this paper, we assume that $\mathrm{Y} = \{\pm 1\}$(e.g. paraphrase/non-paraphrase) for simplicity. Our goal is to predict correct response for a new string re-writing.

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

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

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

$$\phi_r(s,t) = n\lambda^i \tag{2}$$

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 the occurrence of wildcard.

A *re-writing rule* is defined as a triple $r = (\beta_s, \beta_t, \tau)$ where β_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 to an element in the wildcard domain D , and the characters substituting $\beta_s[i]$ and $\beta_t[j]$ are same if 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 controlled by re-writing rule set R and wildcard domain D. For example, pairwise k-spectrum kernel (ps-SRK) citeLes:02 is SRK under $R = \{(\beta_s, \beta_t, \tau) | \beta_s, \beta_t \in \Sigma^k, \tau = \emptyset\}$ and any D. Pairwise k-wildcard kernel (pw-SRK) [Leslie and Kuang, 2004] is SRK under $R = \{(\beta_s, \beta_t, \tau) | \beta_s, \beta_t \in (\Sigma \cup \{*\})^k, \tau = \emptyset\}$ and $D = \Sigma$. Both of them can be decomposed 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.

4 K-gram Bijective String Re-writing Kernel

Next we propose an 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 cannot be decomposed.

4.1 Definition

The kb-SRK has the following properties:

- A wildcard can only substitute a single character, denoted as "?".
- The two string patterns in a re-writing rule are of length k.
- 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.

A kb-SRK is uniquely determined by the re-writing rule set R and the wildcard domain D. 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

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

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

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

where

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

4.2 Algorithm for Computing Kernel

A straightforward computation of kb-SRK would be intractable. In this section, we will introduce an efficient algorithm by example . A detailed explanation can be found in [Bu *et al.*, 2012]. We will first show how to compute $\bar{\mathrm{K}}_k$ (Eq. 5) and then show how to get K_k from $\bar{\mathrm{K}}_k$ (Eq. 4).

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

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: The pair of double lists transformed from the two *k*-gram pairs in Fig. 2. Non-identical doubles are in bold.

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

First, we transform the computation of re-writing rule matching on k-gram pairs into the computation on re-writing rule matching on pairs of double lists. For the two *k*-gram pairs in Fig. 2 , we transform them into the pair of double lists in Fig. 3. Specifically, double (a,a) is from the first positions of α_{s_1} and α_{s_2}, double (b,b) is from the second positions of α_{s_1} and α_{s_2}, and so on. We require each re-writing rule for double lists only consists of identical doubles (i.e. doubles consist of the same characters) and wildcards. It can be shown that this transformation establishes a one-to-one mapping from the set of re-writing rules matching two *k*-gram pairs such as those in Fig. 2 to the set of re-writing rules matching pairs of double lists such as that in Fig. 3.

Now, we just need to compute the number of re-writing rules matching the obtained pair of double lists. Instead of enumerating all possible re-writing rules and checking whether they can match the pair of double lists, we calculate the number of possibilities of "generating" re-writing rules from the pair of double lists, which can be carried out efficiently. It is easy to see that the number of all generations depends only on the number of times each double occurs. Thus, we build a counting vector for each double list, as shown in Fig 5.

In each generation, the identical doubles in the two double lists 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 (to ensure the one-to-one mapping). Next, we will show how to compute $\bar{\mathrm{K}}_k$ of the example in Fig 5. For identical double (c,c), it can only be used to generate 0,1, and 2 pairs of aligned wildcards. When we use it to generate i pairs of aligned wildcards, we select i times of (c,c) from α_s^{D}, i times of (c,c) from α_t^{D} and try all possible alignments. Since each occurrence of a wildcard is penalized by λ, (c,c) contributes $(1+6\lambda^2+6\lambda^4)$. The situations are similar for (a,a) and (b,b). For non-identical double (b,d), since all occurrences of it must be substituted by wildcards, we must use all of them, which results in $2!\lambda^4$. The situation is similar for (b,c). Finally, by the rule of product, we have $\mathrm{K}_k = (1)(1+\lambda^2)(\lambda^2)(2!\lambda^4)(1+6\lambda^2+6\lambda^4)$. It can be proved that this process can be carried out in $O(k)$.

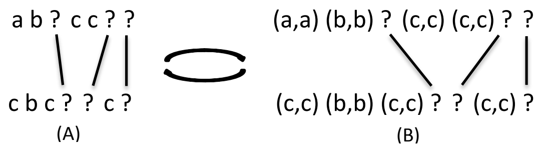

Figure 4: For re-writing rule (A) matching both k-gram pairs shown in Fig. 2, there is a corresponding re-writing rule (B) matching the pair of double lists shown in Fig. 3.

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

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

Computing K_k

To compute K_k in Eq. 4, we need in principle enumerate all possible combinations of α_{s_1}, α_{s_2}, α_{t_1}, α_{t_2}, compute $\bar{\mathrm{K}}_k$ and sum them up. Therefore, a naive implementation K_k would call $\bar{\mathrm{K}}_k$ for $O(n^4)$ times.

Fortunately, not all combinations are necessary to be considered. We prune the search space based on the following two facts.

First, $\bar{\mathrm{K}}_k((\alpha_{s_1},\alpha_{t_1}),(\alpha_{s_2},\alpha_{t_2})) = 0$, if any non-identical double appears different times in α_s^{D} and α_t^{D}, where α_s^{D} and α_t^{D} are composed from α_{s_1}, α_{s_2} and α_{t_1}, α_{t_2}. This is because a re-writing rule for double lists only consists of identical doubles and wildcards and the wildcard alignment within it is bijective. Therefore, we can group together those α_s^{D} and α_t^{D} which have the same occurrences of non-identical doubles and make enumeration inside each group.

Second, $\#_{\Sigma\times\Sigma}(\alpha_s^{\mathrm{D}})$ and $\#_{\Sigma\times\Sigma}(\alpha_t^{\mathrm{D}})$ provide sufficient statistics for computing $\bar{\mathrm{K}}_k((\alpha_{s_1},\alpha_{t_1}),(\alpha_{s_2},\alpha_{t_2}))$. That is to say, we can enumerate α_{s_1}, α_{s_2} and store different $\#_{\Sigma\times\Sigma}(\alpha_s^{\mathrm{D}})$ with their counts. Then we can enumerate α_{t_1}, α_{t_2} and store different $\#_{\Sigma\times\Sigma}(\alpha_t^{\mathrm{D}})$ with their counts and finally compute $\bar{\mathrm{K}}_k$ by enumerating different $\#_{\Sigma\times\Sigma}(\alpha_s^{\mathrm{D}})$ and $\#_{\Sigma\times\Sigma}(\alpha_t^{\mathrm{D}})$.

With these two facts we design an algorithm which calls $\bar{\mathrm{K}}_k$ for only $O(n^2)$ times for computing $\bar{\mathrm{K}}_k$. It first enumerates α_{s_1}, α_{s_2} and groups different $\#_{\Sigma\times\Sigma}(\alpha_s^{\mathrm{D}})$ by their non-identical double statistics. Then it does the same thing on α_{t_1} and α_{t_2}. Finally, it enumerates different $\#_{\Sigma\times\Sigma}(\alpha_s^{\mathrm{D}})$ and $\#_{\Sigma\times\Sigma}(\alpha_t^{\mathrm{D}})$ inside each group and sums them up. The detailed description of the algorithm can be found in [Bu *et al.*, 2012].

5 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.

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

We normalized each kernel by $\tilde{K}(x,y) = \frac{K(x,y)}{\sqrt{K(x,x)K(y,y)}}$ 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 $K_1 + \ldots + K_{k_{max}}$ ($k_{max} \in \{1, 2, 3, 4\}$) together with the combination with PR and we report the best accuracies of them in Table 1 and Table 2.

5.1 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 significantly 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. Note that the recent work in [Socher *et al.*, 2011] achieves 76.8% accuracy on this dataset. However, it uses additional data to train the model, which is not directly comparable to the other methods.

Fig. 6 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 values are all below 71%, and therefore they are not shown for clarity. 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

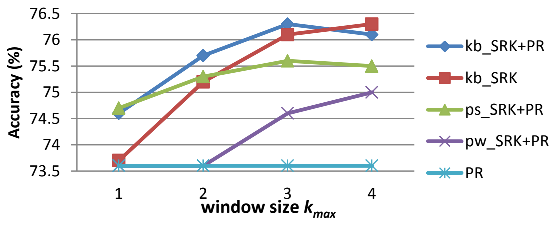

Figure 6: Performances of different kernels under different maximum window size k_{max} on MSRP.

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.

5.2 Recognizing Textual Entailment

Recognizing textual entailment is to determine whether a sentence can entail the other sentence. We combined the development set of RTE-3 and the 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.

6 Conclusion

In this paper, we have proposed a novel class of kernel functions 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 achieves comparable results with state-of-the-art methods on paraphrase identification and recognizing textual entailment.

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.

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.

References

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

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

[Ben-Hur and Noble, 2005] A. Ben-Hur and W.S. Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21:i38–i46, 2005.

[Bu *et al.*, 2012] Fan Bu, Hang Li, and Xiaoyan Zhu. String re-writing kernel. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 449–458, 2012.

[Chang and Lin, 2011] C. Chang and C. Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011.

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

[de Marneffe *et al.*, 2006] M. de Marneffe, B. MacCartney, T. Grenager, D. Cer, Rafferty A., and C.D. Manning. Learning to distinguish valid textual entailments. In *Proc. of the Second PASCAL Challenges Workshop*, pages 468–476, 2006.

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

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

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

[Kashima *et al.*, 2009] H. Kashima, S. Oyama, Y. Yamanishi, and K. Tsuda. On pairwise kernels: An efficient alternative and generalization analysis. *Advances in Knowledge Discovery and Data Mining*, pages 1030–1037, 2009.

[Leslie and Kuang, 2004] C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *The Journal of Machine Learning Research*, 5:1435–1455, 2004.

[Leslie *et al.*, 2002] C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel: a string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, volume 575, pages 564–575, 2002.

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

[Lintean and Rus, 2011] M. Lintean and V. Rus. Dissimilarity kernels for paraphrase identification. In *Proc. of Twenty-Fourth International FLAIRS Conference*, 2011.

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

[MacCartney and Manning, 2008] B. MacCartney and C.D. Manning. Modeling semantic containment and exclusion in natural language inference. In *Proceedings of the 22nd International Conference on Computational Linguistics*, volume 1, pages 521–528, 2008.

[Moschitti and Zanzotto, 2007] A. Moschitti and F.M. Zanzotto. Fast and effective kernels for relational learning from texts. In *Proceedings of the 24th Annual International Conference on Machine Learning*, 2007.

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

[Quirk *et al.*, 2004] C. Quirk, C. Brockett, and W. Dolan. Monolingual machine translation for paraphrase generation. In *Proceedings of EMNLP 2004*, pages 142–149, 2004.

[Socher *et al.*, 2011] Richard Socher, Eric H Huang, Jeffrey Pennington, Andrew Y Ng, and Christopher D Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. *Advances in Neural Information Processing Systems*, 24:801–809, 2011.

[Wan *et al.*, 2006] S. Wan, M. Dras, R. Dale, and C. Paris. Using dependency-based features to take the "para-farce" out of paraphrase. In *Proc. of the Australasian Language Technology Workshop*, pages 131–138, 2006.

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

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