A Convolution Kernel Approach to Identifying Comparisons in Text

Maksim Tkachenko
School of Information Systems
Singapore Management University
maksim.tkatchenko@gmail.com

Hady W. Lauw
School of Information Systems
Singapore Management University
hadywlauw@smu.edu.sg

Abstract

Comparisons in text, such as in online reviews, serve as useful decision aids. In this paper, we focus on the task of identifying whether a comparison exists between a specific pair of entity mentions in a sentence. This formulation is transformative, as previous work only seeks to determine whether a sentence is comparative, which is presumptuous in the event the sentence mentions multiple entities and is comparing only some, not all, of them. Our approach leverages not only lexical features such as salient words, but also structural features expressing the relationships among words and entity mentions. To model these features seamlessly, we rely on a dependency tree representation, and investigate the applicability of a series of tree kernels. This leads to the development of a new context-sensitive tree kernel: Skip-node Kernel (SNK). We further describe both its exact and approximate computations. Through experiments on real-life datasets, we evaluate the effectiveness of our kernel-based approach for comparison identification, as well as the utility of SNK and its approximations.

1 Introduction

When weighing various alternatives, users increasingly turn to the social media, by scouring online reviews, discussion forums, etc. Our goal is to extract from such corpora those text snippets where users make direct comparisons of entities. While sentiment analysis (Pang and Lee, 2008) may be helpful in evaluating individual entities, comparison by the same author within a sentence provides an unambiguous and more equitable basis for the relative positions of two entities on some aspect. For example, the sentence s_1 in Table 1, taken from an Amazon review about a digital camera, makes two distinct comparisons: #1) between "A630" and "A-series cameras" and #2) between "A630" and "its competition", with a clear sense of which entity mention is the *greater* on some aspect ("larger"). Moreover, comparisons may be objective (e.g., larger) or subjective (e.g., better), while sentiments are primarily subjective.

Problem Given a sentence and a specific pair of entity mentions, we seek to determine if a comparison exists between those two mentions. In previous work, the problem was formulated as identifying *comparative sentences*, i.e., those containing at least one comparison (Jindal and Liu, 2006a). This is not ideal because a sentence may contain more than two entity mentions, and may be comparing only some of them. For instance, s_1 is comparative with respect to the pair (A630, A-series cameras) and the pair (A630, its competition), but not the pair (A-series cameras, its competition).

We therefore postulate that the more appropriate formulation is *comparisons within sentences*. If a sentence compares two entities (A, B) with respect to some aspect Z, it should be possible to reformulate it into another sentence such as: "A is better than B with respect to Z" (Kessler and Kuhn, 2014a). Based on this definition, there is no comparison between (A-series cameras, its competition) in s_1. Here, we adopt this apt definition with a slight restriction to make it more practical, and seek to identify such comparisons automatically. We consider only sentences with at least two entity mentions involved in gradable comparisons, i.e., a clear sense of scaling in the comparison (e.g., A is better than B.). Such comparisons are more useful in investigating the pros and cons of entities, as opposed to equative comparisons expressing parity between two mentions (e.g., A is as good as B.), or superlative comparisons expressing the primacy of an entity with respect to unknown reference entities (e.g., A is the best.).

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 376–386,
Beijing, China, July 26-31, 2015.

ID	Sentence	Remarks
s_1	The A630 is slightly larger than previous generation A-series cameras, and also larger than much of its competition.	Contains two comparisons: (A630, A-series cameras) and (A630, its competition).
s_2	I got 30D for my wife because she wanted a better camera.	Includes comparative predicate "better", but contains no comparison.
s_3	I had D3100 and it was nice but the D5100 is truly amazing.	No comparative predicate, but has a comparison: (D3100, D5100).
s_4	D7000 and D7100 do better at high ISO than D300s.	Contains two comparisons: (D7000, D300s) and (D7100, D300s).

Table 1: Example Sentences with ≥ 2 Entity Mentions from Amazon.com Digital Cameras Reviews

Approach For English, there usually is a comparative predicate that anchors a comparison, such as "better" or "worse". However, many sentences with such predicate words are not comparisons. The sentence s_2 in Table 1 has the word "better", but does not contain any comparison between the entity mentions. Yet, other words (e.g., "amazing"), though not a comparative predicate, could signify a comparison, e.g., in s_3 in Table 1.

(Jindal and Liu, 2006a) considered the "context" around a predicate. A sentence is transformed into a sequence involving the predicate and the part of speech (POS) within a text window around the predicate (usually three words before and after). For instance, s_2 in Table 1 would be transformed into the sequence $\langle$*PRP VBD DT better NN*$\rangle$. Such sequences are labeled comparative or non-comparative, upon which (Jindal and Liu, 2006a) applies sequential pattern mining (Agrawal and Srikant, 1995; Ayres et al., 2002; Pei et al., 2001) to learn *class sequential rule* (CSR). These CSRs are then used as features in classifying comparative sentences.

While (Jindal and Liu, 2006a) makes some progress by considering context, its performance may be affected by several factors. First, CSRs are not sensitive to entity mentions. It may classify s_1 as comparative generally, missing the nuance that s_1 is not comparing the pair (A-series cameras, its competition). Second, as CSRs requires a list of comparative predicates, the quality and the completeness of the list are crucial. For instance, "amazing" is not in their list, and thus the comparison in s_3 may not be identifiable. Third, due to the windowing effect, CSRs has a limited ability to model long-range dependencies. For s_4, a window of three words around the predicate "better" excludes the word "than" that would have been very informative. Yet, enlarging the window might then bring in irrelevant associations.

What is important then is not so much whether a sentence is comparative as whether two entity mentions are related by a comparative relation. One insight we draw is how comparison identification is effectively a form of *relation extraction*. While there are diverse relation extraction formulations (Culotta and Sorensen, 2004; Bunescu and Mooney, 2005; Nguyen et al., 2009), our distinct relation type is comparison of two entity mentions.

Armed with this insight, we propose a kernel-based approach based on a dependency tree representation (Nivre, 2005), with significant innovations motivated by the comparative identification task. This proposed approach has several advantages over CSR. Most importantly, it models dependencies between any pair of words (including entity mentions), whereas CSR only relates a comparative predicate to nearby POS tags. For other advantages, unlike CSR, this approach is contingent on neither a pre-specified list of comparative predicates, nor a specific window length.

Contributions In this paper, we make the following contributions. *First*, we re-formulate the problem of automatic identification of comparative sentences into the more general task of identifying comparisons within sentences. *Second*, we propose to frame comparison identification as a relation extraction problem. This entails: #1) deriving an appropriate dependency tree representation of sentences to enable discrimination of comparison vs. non-comparison within the same sentence (see Section 2), and #2) a systematic exploration of the applicability of various tree kernel spaces to our task (see Section 3). *Third*, due to the limitation of the existing tree kernels, we propose a new tree kernel: Skip-node Kernel that is context-sensitive, and discuss both its exact and approximate computations (see Section 4). *Fourth*, we validate its effectiveness and efficiency through experiments on real-life datasets (see Section 5).

2 Overview

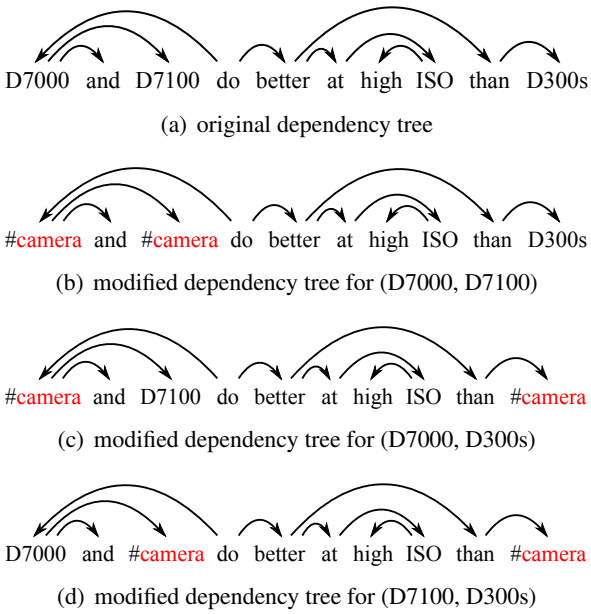
Task The input is a corpus of sentences $\mathcal{S}$ concerning a set of entities within a certain domain (e.g., digital cameras). Every sentence $s \in \mathcal{S}$ contains at least two entity mentions. The set of entity mentions in s is denoted M_s. For instance, the sentence s_4 in Table 1 contains three entity mentions: D7000, D7100, and D300s. The same entity may be mentioned more than once in a sentence, in which case every mention is a distinct instance.

As output, we seek to determine, for each pair of entity mentions $(m_i < m_j) \in M_s$ in a sentence $s \in \mathcal{S}$, a binary class label of whether s contains a comparison between m_i and m_j. For the pair (D7000, D7100) in s_4, the correct class is 0 (no comparison). For the other two pairs (D7000, D300s) and (D7100, D300s), the correct class is 1 (comparisons). We do not seek to identify the aspect of comparison, which is a different problem of independent research interest (see Section 6).

Dependency Tree In order to represent both the lexical units (words) as well their structural dependencies seamlessly, we represent each sentence s as a dependency tree T. For example, Figure 1(a) shows the dependency tree of s_4 in Table 1. The tree is rooted at the main verb ("do"), and each dependency relation associates a head word and a dependent word. To describe a tree or any of its substructures, we use the bracket notation. Figure 1(a) in this notation is `[do [D7000 [and] [D7100]] [better [at [ISO [high]]] [than [D300s]]]]`.

Here, we make two observations. First, there is one tree even for a sentence with multiple pairs of entity mentions. Second, the information signalling a comparison is borne by the structures around the mentions (e.g., `[better [than]]`, rather than the actual mentions (e.g., "D7000"). These lead us to introduce a *modified* dependency tree that is distinct for every pair of mentions, achieved by replacing each entity mention of interest by a placeholder token. Here, we use the token "#camera" for illustration. Figure 1(b) shows the modified tree for the pair (D7000, D7100). This enables learning in an entity-agnostic way, because the token ensures that sentences about different cameras are interpreted similarly.

Convolution Kernel Observe how the trees of the pair (D7000, D300s) in Figure 1(c) and the pair (D7100, D300s) in Figure 1(d), which are both comparisons, share certain substructures, such as `[do [better [than [#camera]]]]`. In contrast, the tree in Figure 1(b) for the pair (D7000, D7100), which is not a comparison, does not contain this substructure. What we need is a way to systematically examine tree substructures to determine the similarity between two trees.



(a) original dependency tree

(b) modified dependency tree for (D7000, D7100)

(c) modified dependency tree for (D7000, D300s)

(d) modified dependency tree for (D7100, D300s)

Figure 1: Modified dependency trees.

Kernel methods offer a way to measure the similarity by exploring an implicit feature space without enumerating all substructures explicitly. Suppose that $\mathbf{T}$ denotes the space of all possible instances. A kernel function K is a symmetric and positive semidefinite function that maps the instance space $\mathbf{T} \times \mathbf{T}$ to a real value in the range of $[0, \infty)$ (Haussler, 1999). A tree kernel function can be reformulated into a *convolution kernel* (Collins and Duffy, 2001), shown in Equation 1.

$$K(T_1, T_2) = \sum_{n_i \in T_1} \sum_{n_j \in T_2} D(n_i, n_j) \qquad (1)$$

Here, n_i and n_j denote each node in their respective tree instances T_1 and T_2. $D(n_i, n_j)$ is the number of common substructure instances between the two sub-trees rooted in n_i and n_j respectively. The exact form of $D(n_i, n_j)$ depends on the specific definition of the tree kernel space. In Section 3, we systematically explore the applicability of various tree kernel spaces, leading to the introduction of the new *Skip-node Kernel*.

The appropriate kernel function can be embedded seamlessly in kernel methods for classification. In this work, we use the Support Vector Machines (SVM) (Steinwart and Christmann, 2008).

3 Tree Kernel Spaces

Tree kernels count substructures of a tree in some high-dimensional feature space. Different tree kernel spaces vary in the amount and the type of information they can capture, and thus may suit different purposes. To find a suitable tree kernel for the comparison identification task, we first systematically explore a progression of known tree kernel spaces, including Sub-tree, Subset Tree, and Partial Tree. Through the use of appropriate examples, we show how these existing tree kernel spaces may not be appropriate for certain instances. This section culminates in the introduction of a new feature space that we call Skip-node.

Sub-tree (ST) Space In this space, the basic substructure is a subgraph formed by a node along with all its descendants. Applying this kernel to two dependency trees of similar sentences may not be appropriate due to, for example, modifier words that change the dependency structure. To illustrate this, let us examine the two dependency parses in Figure 2. Both support comparisons, and ideally we can detect some level of similarity. However, if we consider only sub-trees, the two dependency trees share in common only two fragments: `[#camera]` and `[is]`. Neither of these fragments is indicative of a comparison.

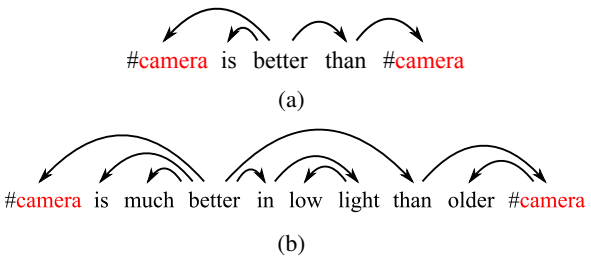


Figure 2: Dependency parses. Working example for the Sub-tree, Subset Tree, Partial Tree kernels.

Subset Tree (SST) Space We next consider the SST kernel, which computes similarity in a more general space of substructures than ST. Any subgraph of a tree that preserves production rules is counted. This definition suggests SST is intended more for a constituency parse (Moschitti, 2006a). In this feature space, the parses in Figure 2 now have in common the following fragments: `[#camera]`, `[is]`, `[than [#camera]]`. This representation is better than ST's, e.g., the fragment `[than [#camera]]` is informative. However, as a whole, the set of features are still insufficient to identify a comparison.

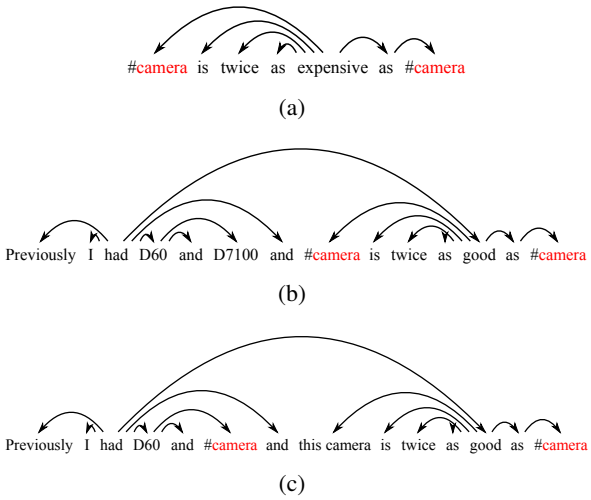


Figure 3: Dependency parses. Working example for the Partial Tree, Skip-node kernels.

Partial Tree (PT) Space In turn, the PT space allows breaking of production rules, making it a better choice than SST for dependency parses. PT kernel would find that the parse in Figure 2(a) with all its subgraphs can be matched as a whole within the parse in Figure 2(b), identifying a close match.

However, PT kernel is prone to two drawbacks. By generating an exponential feature space, it may overfit and degrade generalization (Cumby and Roth, 2003). More importantly, PT considers tree fragments independently from their contexts, resulting in features involving non-related parts of a sentence. This is particularly apparent when we consider multiple entities within a sentence.

Suppose that Figure 3(a) is in our training set, and we have the sentence below in the testing set:

> Previously, I had D60 and D7100, and this camera is twice as good as D60.

Figure 3(b) shows the parse for (this camera, D60), and Figure 3(c) for (D7100, D60). The former is a comparison, and should match Figure 3(a). The latter is not and should not match. PT kernel cannot resolve this ambiguity, computing the same similarity value to Figure 3(a) for both. The common features are: `[#camera]`, `[is]`, `[twice]`, `[as]`, and `[as [#camera]]`.

Skip-node (SN) Space Figures 3(a) and 3(b) share a similar substructure "twice as ... as", but because they use different words to express the comparisons ("expensive" vs. "good"), previous kernels treat their features disjointly, missing out on their similarity. To reduce this over-reliance on exact word similarity, we seek a feature space that

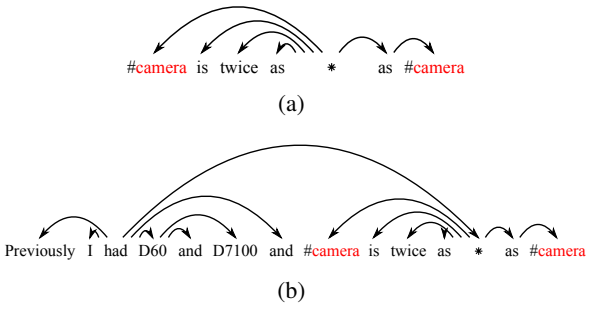


Figure 4: Dependency parses with skipped nodes.

would allow some degree of relaxation in determining the *structural* similarity between trees.

We therefore propose the Skip-node (SN) space, which represents a generalized space of tree fragments, where some nodes can be "skipped" or relabeled to a special symbol '$\star$' that would match nodes of any label. A restriction on this space is that each skip symbol must connect two non-skip (regular) nodes. The implication is that skips code for some notion of connecting distance between non-skip nodes. Moreover, the space would not include features such as `[* [* [#camera]]]` that serve only to indicate the presence of ancestors, and not any relationship of non-skip nodes.

Figure 4 resolves the ambiguity in Figure 3 by skipping the words "expensive" and "good", introducing a new set of features: `[* [#camera] [is] [twice] [as] [as [#camera]]]`. Note how in this case the skip symbol effectively serves as a "context" that pulls together the previously disjoint features identified by the PT kernel. These new context-sensitive features would allow a match between the earlier Figures 3(a) and 3(b), but not Figure 3(c).

Thus, SN space effectively generalizes over the PT space, and enriches it with context-sensitive features. To avoid overfitting, in addition to decay parameter λ used in PT kernel, we associate SN kernel with two other parameters. The SN space consists of rooted ordered trees where some nodes are labeled with a special skip symbol '$\star$', such that the number of regular nodes (not marked with '$\star$') is at most S, and each skip node is within a distance of L from a non-skip node. This engenders a graceful gradation of similarity as the number of skip nodes in a substructure grows, yet imposes a limit to the extent of relaxation.

4 Skip-node Kernel Computation

We now discuss the computation of Skip-node Kernel, first exactly, and thereafter approximately.

4.1 Exact Computation

We define the alignment of common fragments between two trees in the Skip-node space. When $S = 1$, only singleton nodes with the same labels contribute to the kernel, and alignment is straightforward. When aligning fragments with two regular nodes ($S > 1$), we consider their connection structure and the order of the child nodes to prevent over-counting substructures with the same labels (e.g., `[*[as][as]]` in Figure 4). To preserve the natural order of words in a sentence, we enumerate the tree nodes according to preorder, left-to-right depth-first search (DFS) traversal.

In turn, the connection structure is defined by the skip-node path connecting two regular nodes. This can be expressed as a sequence of upward (towards the root) and downward (towards the leaves) steps we need to perform to get from the leftmost to the rightmost regular node. Due to the natural ordering of regular nodes, upward steps are followed by downward steps. The sequence can be expressed as a pair of numbers: $\langle \rho(n_l, u), \rho(n_r, u) \rangle$, where n_l is the leftmost regular node of a fragment, n_r is the rightmost one, $u = \sigma(n_l, n_r)$ is the lowest common ancestor of nodes n_l, n_r, and ρ returns the number of edges in the shortest path connecting two nodes.

Suppose a rooted tree $T = (N, E)$ has preorder DFS enumeration $N = (n_1, n_2, ..., n_{|N|})$. For $i < j$, we define a function $\pi(n_i, n_j)$, which canonically represents the way two nodes are connected in a tree, as follows:

$$\pi(n_i, n_j) = \langle \rho(n_i, \sigma(n_i, n_j)), \rho(n_j, \sigma(n_i, n_j)) \rangle.$$

DEFINITION 1 (STRUCTURAL ISOMORPHISM): Given two trees $T_1 = (N_1, E_1)$, $T_2 = (N_2, E_2)$, we say that pairs of nodes $(v_i, u_{i'}), (v_j, u_{j'}) \in N_1 \times N_2$ are *structurally isomorphic* and write $(v_i, u_{i'}) \leftrightsquigarrow (v_j, u_{j'})$ when $\pi(v_i, v_j) = \pi(u_{i'}, u_{j'})$ on the valid domain.

It can be shown that structural isomorphism is a transitive relation. This property allows us to grow aligned fragments by adding one node at a time:

$$(v_i, u_{i'}) \leftrightsquigarrow (v_j, u_{j'}) \wedge (v_j, v_{j'}) \leftrightsquigarrow (v_k, u_{k'}) \Rightarrow (v_i, u_{i'}) \leftrightsquigarrow (v_k, u_{k'}).$$

To compute the kernel, we use a graph-based approach to enumerate all the common substructures in the Skip-node space. Given two trees T_1 and T_2, we begin by aligning their nodes. The sets of nodes in T_1 and T_2 are N_1 and N_2 respectively. Let N_G be a set of pairs $(n_i, n_j) \in N_1 \times N_2$, where n_i and n_j have the same label. On top of N_G, we build a graph $G = (N_G, E_G)$. We draw an edge between two vertices $(v_i, v_k), (u_j, u_l) \in N_G$, if $(v_i, u_j) \leftrightsquigarrow (v_k, u_l)$ and $\rho(v_i, v_k) \leq L$.

Any connected subgraph of G represents a feature in the Skip-node space common to both T_1 and T_2. The kernel then needs to count the number of connected subgraphs of sizes not more than S. To see that this procedure is correct, we simply need to trace back the construction of graph G, and build an bijection from a subgraph of G to the corresponding fragments of T_1 and T_2.

Enumerating all the connected subgraphs of a given graph requires exponential time. The algorithm described above requires $\mathcal{O}(|N_1||N_2| + \sum_{i=1}^{S} \binom{|N_G|}{i})$ time, assuming that the distance between two nodes in a tree can be computed in $\mathcal{O}(1)$ with appropriate linear preprocessing. See (Bender and Farach-Colton, 2000) for insight. The exact computation is still tractable on the condition that S and L are not very large. This condition would probably hold in most realistic scenarios. Yet, to improve the practicality of the kernel, we propose a couple of approximations as follows.

4.2 Approximate Computation

One reason for the complexity of the Skip-node kernel is that although the graph G is formed by aligning two trees, by allowing connections through skips, G itself may not necessarily be in the form of a tree. In deriving an approximation, our strategy is to form G through alignment of linear substructures of the original two trees. A Skip-node space over linear structures can be computed in polynomial time using dynamic programming.

Linear Skip-node One approximation is to consider linear substructures in the form of root-paths. A root-path is a path from the root of a tree to a leaf. Given two trees T_1 and T_2, with DFS enumerated nodes $N_1 = (v_1, v_2, ..., v_{m_1})$ and $N_2 = (u_1, u_2, ..., u_{m_2})$ respectively. Here, v_1 and u_1 are roots, and v_{m_1} and u_{m_2} are the leaves. Starting with common fragments at the leaves, we grow them into larger common fragments towards the root. We call this approximation *Linear Skip-node*. Figure 5(a) shows examples of features considered by Linear Skip-node for the illustrated tree T in skip-node space ($S = 3, L = 2$).

The kernel function can be decomposed into:

$$K(T_1, T_2) = \sum_{v_i \in N_1} \sum_{u_j \in N_2} \sum_{s=1}^{S} \lambda^s D(v_i, u_j, s),$$

where $D(v_i, u_j, s)$ is the number of common substructures of size s with the leftmost regular nodes v_i and u_j. λ is a decay factor for substructure size.

The recursive definition of the kernel is:

$$D(v_i, u_j, s) = \sum_{i<k\leq m_1} \sum_{j<l\leq m_2} I(v_i, v_k, u_j, u_l) D(v_k, u_l, s-1),$$

$$D(v_i, u_j, 1) = \begin{cases} 1 & \text{if } label(v_i) = label(u_j), \\ 0 & \text{otherwise;} \end{cases}$$

$$I(v_i, v_k, u_j, u_l) = \mathbb{1}_{(v_i, u_j) \leftrightsquigarrow (v_k, u_l)} \mathbb{1}_{\rho(v_i, v_k) \leq L} \cdot \mathbb{1}_{(v_i \text{ is an ancestor of } v_k)},$$

where $\mathbb{1}_c$ equals 1 when constraint c is satisfied and 0 otherwise. Note that the first two factors of indicator function I just represent the general Skip-node space constraints, the last factor ensures that features are computed along the root-paths.

Lookahead Skip-node The second approximation, *Lookahead Skip-node*, is related to the observation that when growing a substructure, we do not have to confine the growth only towards ancestors, as DFS traversal already ensures iterative manner of computation. In other words, the constraint v_i is an ancestor of v_k can be dropped:

$$I(v_i, v_k, u_j, u_l) = \mathbb{1}_{(v_i, u_j) \leftrightsquigarrow (v_k, u_l)} \cdot \mathbb{1}_{\rho(v_i, v_k) \leq L}.$$

In addition to those features generated by Linear Skip-node in Figure 5(a), Lookahead Skip-node can generate additional tree substructures, shown in Figure 5(b). The approximation can be computed using different DFS enumerations, which may result in different feature sets. In our experiments, we used pre-order left-to-right enumeration. Given the enumeration of tree T as in Figure 5, we start to grow feature fragments from node n_4. According to the Skip-node space constraints, the growth can only proceed to nodes n_1 or n_2. Once any of these nodes is attached to n_4, we lose tree fragments containing n_3, as the procedure allows us to grow substructures only towards nodes with smaller (earlier) DFS enumer-

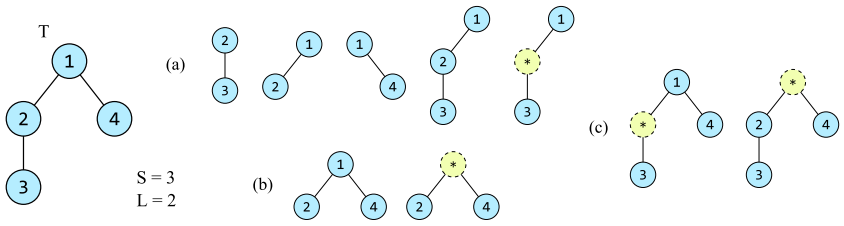


Figure 5: Features of T in skip-node space $(S = 3, L = 2)$. Numbers indicates pre-order left-to-right DFS enumeration of T. Dashed circles represent skip nodes. Subfigures: (a) - modeled by all; (b) - modeled by Lookahead Skip-node, not by Linear Skip-node; (c) - modeled only by Exact Skip-node.

Domain	# sentences	% comp.	# pairs	% comp.
Camera	1716	59.4%	2170	49.9%
Cell	821	35.2%	1110	30.5%

Table 2: The dataset size for each domain.

ation numbers. Figure 5(c) shows the fragments that Lookahead Skip-node cannot capture[1].

The computation procedure is similar for both approximations and requires $\mathcal{O}(S|N_1|^2|N_2|^2)$.

5 Experiments

Data For experiments, we compiled two annotated datasets in two domains: Digital Camera and Cell Phone from online review sentences. The reviews were collected from *Amazon* and *Epinions*[2].

We identified the entity mentions through dictionary matching, followed by manual annotation to weed out false positives. Each dictionary entry is a product name (e.g., *Canon PowerShot D20*, *D7100*) or a common product reference (e.g., *this camera*, *that phone*). The dataset includes only sentences that contain at least two entity mentions. Every pair of entities within a sentence was annotated with a comparative label according to the definition given in Section 2. A sentence is comparative if at least one pair of entities within it is in a comparative relation. Table 2 shows the dataset properties, in terms of the number sentences and the percentage that are comparative sentences, as well as the number of pairs of entity mentions and the percentage that are comparative relations. There are more pairs than sentences, i.e., many sentences mention more than two entities.

This dataset subsumes the annotated gradable comparisons of (Kessler and Kuhn, 2014a) derived from *Epinions* reviews on Digital Cameras. (Jindal and Liu, 2006a)'s dataset is inapplicable, due to its lack of entity-centric comparison.

	Camera			Cell		
	P	R	F1	P	R	F1
CSR	74.3	52.3	61.3	48.9	**61.5***	54.3
BoW	76.9	**76.3**	76.6	62.2	58.0	59.8
BoW†	77.3	71.9	74.4	69.0	56.3	61.8
SNK	**80.5***	75.2	**77.7****	**77.2***	55.1	**64.1***

Table 3: Comparison identification task

Evaluation The experiments were carried out with SVM-light-TK framework[3] (Joachims, 1999; Moschitti, 2006b), into which we built Skip-node Kernel. We further release a separate standalone library that we built, called Tree-SVM[4], which does SVM optimization using the tree kernels described in this paper. The sentences were parsed and lemmatized with the use of the Stanford NLP software (Chen and Manning, 2014).

The experiments were done on 10 random data splits in 80:20 proportion of training vs. testing. Performance is measured by using F_1, which is the harmonic mean of precision P and recall R: $F_1 = \frac{2PR}{P+R}$. The statistical significance[5] is measured by randomization test (Yeh, 2000). The hyper-parameters, including the baselines', were optimized for F_1 through grid-search.

5.1 Comparison Identification

Our first and primary objective is to investigate the effectiveness of the proposed approach on the task of identifying comparisons between a pair of en-

[1] In this particular case, all features could have been computed by Lookahead Skip-node using preorder right-to-left DFS enumeration, although it may not be true in general.

[2] We used already available snapshots for *Epinions* dataset: http://groups.csail.mit.edu/rbg/code/precis/.

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

[4] http://github.com/sitfoxfly/tree-svm

[5] When presenting the results, an asterisk indicates that the outperformance over the second-best result is significant at 0.05 level. Two asterisks indicate the same at 0.1 level.

	Camera			Cell		
	P	R	F1	P	R	F1
CSR	74.6	51.7	60.9	50.9	**61.2***	55.3
BoW	77.5	**76.3**	76.8	63.4	57.7	60.2
BoW†	77.6	72.4	74.9	70.9	57.3	63.2
SNK	**81.0***	75.2	**78.0****	**77.9***	54.8	**64.2**

Table 4: Comparative sentence identification task

tity mentions. Previous work focused on identifying comparative sentences. We compare to three baselines. One is CSR, implemented following the description in (Jindal and Liu, 2006a). Another is BoW, classification using bag-of-words as features. For the baselines, if a comparative sentence contains more than one pair of entities, we assume that every pair is in comparative relation. The third baseline, BoW†, considers only the words in between of the two target entities.

Table 3 shows the performance on the comparison identification task (best results are in bold). In terms of F_1, it is evident that SNK outperforms the baselines. This is achieved through significant gains in precision. It is expected that the baselines tend to have a high recall. CSR benefits from the human-constructed predefined list of comparative keywords and key phrases that a kernel-based method is unable to learn from a training split. BoW† tends to have a higher precision than the other baselines, as it is able to distinguish between different pairs of entities within one sentence.

While SNK may have an inherent advantage over CSR or BoW due to its entity orientation, to investigate the effectiveness of the method itself, we now compare them on the previous task of comparative sentence identification. Table 4 shows that even in this task, SNK still performs better than the baselines. Comparing Table 3 and Table 4, the results also concur with the intuition: once we fold up multiple entity pairs in a sentence into a comparative sentence, we observe a drop in recall and an increase in precision.

5.2 Tree Kernel Spaces

Our second objective is to explore the progression of feature spaces discussed in Section 3. Table 5 reports the results on comparison identification task. The F_1 columns show that the performance gradually increases from STK to SNK along with the increase in the complexity of feature space. PTK and SNK can be considered high-

	Camera			Cell		
	P	R	F1	P	R	F1
STK	67.5	64.0	64.9	43.7	41.9	42.6
SSTK	72.1	72.6	71.8	**79.6**	42.4	54.9
PTK	79.2	74.9	76.9	72.3	**56.0****	62.7
SNK	**80.5***	**75.2**	**77.7****	77.2	55.1	**64.1***

Table 5: Tree kernels

	Camera			Cell		
	P	R	F1	P	R	F1
STK_{BoW}	79.9	65.1	71.7	**77.5**	45.3	56.8
$SSTK_{BoW}$	78.0	73.5	75.6	71.8	54.5	61.6
PTK_{BoW}	78.6	74.1	76.2	71.0	53.8	60.8
SNK	**80.5**	**75.2****	**77.7***	77.2	**55.1**	**64.1****

Table 6: Tree kernels combined with bag-of-words

variance estimators due to the power of their feature spaces. The data is such that these kernels may not have fully modeled the feature space completely enough to show even sharper differences.

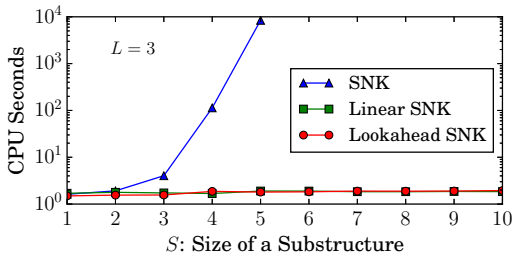
SNK's parameters were optimized to non-trivial cases ($S > 1$ and $L > 1$) by the grid-search, i.e., $S = 3$ and $L = 2$ for Digital Camera and $S = 2$ and $L = 3$ for Cell Phone. The trivial case $S = 1$ represents a standard bag-of-words feature space, i.e., this space is embedded into Skip-node space whenever $S > 1$. To show that SNK does not merely take advantage of this simple space to compete with structural kernels, we carried out another experiment where we combined STK, SSTK, and PTK with bag-of-word representation of a sentence. Table 6 shows that surprisingly this combination harms the quality of PTK. STK and SSTK gain more from bag-of-words features. Nevertheless, the overall outperformance by SNK remains.

5.3 Skip-node Kernel Approximations

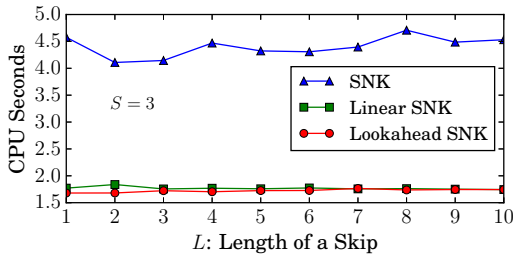
Our third objective is to study the utility of the approximations of SNK described in Section 4. Table 7 reports the performance of the approximations. For Camera, the performance of Lookahead

	Camera			Cell		
	P	R	F1	P	R	F1
Linear SNK	78.9	**77.1***	**77.9**	71.8	**55.3**	62.2
Lookahead SNK	**80.5**	75.2	77.7	71.8	**55.3**	62.2
SNK	**80.5**	75.2	77.7	**77.2***	55.1	**64.1**

Table 7: Effectiveness: SNK vs. approximations



(a) $L = 3, S \in 1..10$



(b) $S = 3, L \in 1..10$

Figure 6: Efficiency: SNK vs. approximations

SNK and SNK are the same. In turn, Linear SNK represents more restricted features, yielding a drop in precision and a gain in recall, resulting in the best F_1. For Cell Phone, the approximations are close, but the original SNK has the best F_1.

To study the running time, we randomly select 500 sentences. Figure 6 shows the time for applying a kernel function to 250k pairs of sentences when we vary two parameters: S and L. When S varies, SNK running time has exponential behaviour, whereas the approximations show fairly linear curves. L seems to influence the computation time linearly for SNK and and its approximations. The experiments were carried out on a PC with Intel Core i5 CPU 3.2 GHz and 4Gb RAM.

This experiment shows that the original SNK is still tractable for small S and L, which turn out to be the case for optimal effectiveness. If efficiency is of paramount importance, the two approximations are significantly faster, without much degradation (none in some cases) of effectiveness.

6 Related Work

Exploiting comparisons in text begins with identifying comparisons within sentences. The previous state of the art for English is the baseline CSR approach (Jindal and Liu, 2006a). For scientific text, (Park and Blake, 2012) explored handcrafted syntactic rules that might not cross domains well. Comparisons are also studied in other languages, such as Chinese, Japanese, and Korean (Huang et al., 2008; Yang and Ko, 2009; Kurashima et al., 2008; Yang and Ko, 2009; Zhang and Jin, 2012).

A different task seeks to identify the "components" within comparative sentences, i.e., entities, aspect, comparative predicate (Jindal and Liu, 2006b; Hou and Li, 2008; Kessler and Kuhn, 2014b; Kessler and Kuhn, 2013; Feldman et al., 2007). Others are interested in yet another task to identify the direction of the comparisons (Ganapathibhotla and Liu, 2008; Tkachenko and Lauw, 2014), or the aggregated ranking (Kurashima et al., 2008; Zhang et al., 2013; Li et al., 2011). Our task precedes these tasks in the pipeline.

Other than comparison identification, dependency grammar has also found applications in natural language-related tasks, such as sentiment classification (Nakagawa et al., 2010), question answering (Punyakanok et al., 2004; Lin and Pantel, 2001), as well as relation extraction (Culotta and Sorensen, 2004; Bunescu and Mooney, 2005).

(Collins and Duffy, 2001) applied convolution kernels (Haussler, 1999; Watkins, 1999) to natural language objects, which evolved into tree kernels, e.g., sub-tree (Vishwanathan and Smola, 2004), subset tree (Collins and Duffy, 2002), descending-path kernel (Lin et al., 2014), partial tree (Moschitti, 2006a). Skip-node kernel joins the list of tree kernels applicable to dependency trees. These kernels may also apply to other types of trees, e.g., constituency trees (Zhou et al., 2007).

(Croce et al., 2011; Srivastava et al., 2013) proposed to capture semantic information along with tree structure, by allowing soft label matching via lexical similarity over distributional word representation. Skip-node gives another perspective on sparsity, using structural alignment of the tree fragments with non-matching labels. As lexical similarity can be incorporated into Skip-node kernel, we consider it orthogonal and complementary.

7 Conclusion

We study the effectiveness of a convolution kernel approach for the novel formulation of extracting comparisons within sentences. Our approach outperforms the baselines in identifying comparisons and comparative sentences. Skip-node kernel and its approximations are particularly effective for comparison identification, and potentially applicable to other relation extraction or natural-language tasks (the direction of our future work).

References

Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining sequential patterns. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 3–14.

Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. 2002. Sequential pattern mining using a bitmap representation. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 429–435.

Michael A. Bender and Martin Farach-Colton. 2000. The lca problem revisited. In *Proceedings of the Latin American Symposium on Theoretical Informatics*, LATIN '00, pages 88–94, London, UK, UK. Springer-Verlag.

Razvan C. Bunescu and Raymond 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 (HLT)*, pages 724–731.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750.

Michael Collins and Nigel Duffy. 2001. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems (NIPS)*, pages 625–632.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the Annual Meeting on Association for Computational Linguistics (COLING)*, pages 263–270.

Danilo Croce, Alessandro Moschitti, and Roberto Basili. 2011. Structured lexical similarity via convolution kernels on dependency trees. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1034–1046. Association for Computational Linguistics.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the Annual Meeting on Association for Computational Linguistics (COLING)*.

Chad Cumby and Dan Roth. 2003. On kernel methods for relational learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 107–114.

Ronen Feldman, Moshe Fresko, Jacob Goldenberg, Oded Netzer, and Lyle Ungar. 2007. Extracting product comparisons from discussion boards. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 469–474.

Murthy Ganapathibhotla and Bing Liu. 2008. Mining opinions in comparative sentences. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pages 241–248.

David Haussler. 1999. Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of California at Santa Cruz.

Feng Hou and Guo-Hui Li. 2008. Mining Chinese comparative sentences by semantic role labeling. In *International Conference on Machine Learning and Cybernetics*, volume 5, pages 2563–2568.

Xiaojiang Huang, Xiaojun Wan, Jianwu Yang, and Jianguo Xiao. 2008. Learning to identify comparative sentences in Chinese text. In *Pacific Rim International Conference on Artificial Intelligence*, pages 187–198.

Nitin Jindal and Bing Liu. 2006a. Identifying comparative sentences in text documents. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 244–251.

Nitin Jindal and Bing Liu. 2006b. Mining comparative sentences and relations. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 22, pages 1331–1336.

Thorsten Joachims. 1999. Advances in kernel methods. chapter Making Large-scale Support Vector Machine Learning Practical, pages 169–184. MIT Press, Cambridge, MA, USA.

Wiltrud Kessler and Jonas Kuhn. 2013. Detection of product comparisons-how far does an out-of-the-box semantic role labeling system take you? In *Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP)*, pages 1892–1897.

Wiltrud Kessler and Jonas Kuhn. 2014a. A corpus of comparisons in product reviews. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, may.

Wiltrud Kessler and Jonas Kuhn. 2014b. Detecting comparative sentiment expressions – a case study in annotation design decisions. In *Proceedings of Konferenz zur Verarbeitung Natrlicher Sprache (KONVENS)*, October.

Takeshi Kurashima, Katsuji Bessho, Hiroyuki Toda, Toshio Uchiyama, and Ryoji Kataoka. 2008. Ranking entities using comparative relations. In *Database and Expert Systems Applications (DEXA)*, pages 124–133.

Si Li, Zheng-Jun Zha, Zhaoyan Ming, Meng Wang, Tat-Seng Chua, Jun Guo, and Weiran Xu. 2011. Product comparison using comparative relations. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 1151–1152.

Dekang Lin and Patrick Pantel. 2001. Discovery of inference rules for question-answering. *Natural Language Engineering*, 7(04):343–360.

Chen Lin, Timothy Miller, Alvin Kho, Steven Bethard, Dmitriy Dligach, Sameer Pradhan, and Guergana Savova. 2014. Descending-path convolution kernel for syntactic structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 81–86. Association for Computational Linguistics.

Alessandro Moschitti. 2006a. Efficient convolution kernels for dependency and constituent syntactic trees. In *European Conference on Machine Learning (ECML)*, pages 318–329.

Alessandro Moschitti. 2006b. Making tree kernels practical for natural language learning. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 113–120.

Tetsuji Nakagawa, Kentaro Inui, and Sadao Kurohashi. 2010. Dependency tree-based sentiment classification using crfs with hidden variables. In *Human Language Technologies: Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 786–794.

Truc-Vien T Nguyen, Alessandro Moschitti, and Giuseppe Riccardi. 2009. Convolution kernels on constituent, dependency and sequential structures for relation extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1378–1387.

Joakim Nivre. 2005. Dependency grammar and dependency parsing. *MSI report*, 5133(1959):1–32.

Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135.

Dae Hoon Park and Catherine Blake. 2012. Identifying comparative claim sentences in full-text scientific articles. In *Proceedings of the Workshop on Detecting Structure in Scholarly Discourse*, pages 1–9.

Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 2001. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 0215–0215.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2004. Natural language inference via dependency tree mapping: An application to question answering. *Computational Linguistics*, 6(9).

Shashank Srivastava, Dirk Hovy, and Eduard Hovy. 2013. A walk-based semantically enriched tree kernel over distributed word representations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1411–1416. Association for Computational Linguistics.

Ingo Steinwart and Andreas Christmann. 2008. *Support vector machines*. Springer.

Maksim Tkachenko and Hady W Lauw. 2014. Generative modeling of entity comparisons in text. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 859–868.

SVN Vishwanathan and Alexander Johannes Smola. 2004. Fast kernels for string and tree matching. *Kernel Methods in Computational Biology*, pages 113–130.

Chris Watkins. 1999. Dynamic alignment kernels. *Advances in Neural Information Processing Systems (NIPS)*, pages 39–50.

Seon Yang and Youngjoong Ko. 2009. Extracting comparative sentences from Korean text documents using comparative lexical patterns and machine learning techniques. In *Proceedings of the ACL-IJCNLP Conference Short Papers*, pages 153–156.

Alexander Yeh. 2000. More accurate tests for the statistical significance of result differences. In *Proceedings of the Conference on Computational Linguistics (COLING)*, pages 947–953. Association for Computational Linguistics.

Runxiang Zhang and Yaohong Jin. 2012. Identification and transformation of comparative sentences in patent Chinese-English machine translation. In *International Conference on Asian Language Processing (IALP)*, pages 217–220.

Zhu Zhang, Chenhui Guo, and Paulo Goes. 2013. Product comparison networks for competitive analysis of online word-of-mouth. *ACM Transactions on Management Information Systems (TMIS)*, 3(4):20.

GuoDong Zhou, Min Zhang, Dong Hong Ji, and Qiaoming Zhu. 2007. Tree kernel-based relation extraction with context-sensitive structured parse tree information. *EMNLP-CoNLL*, page 728.