Labeling by Landscaping: Classifying Tokens in Context by Pruning and Decorating Trees

Siddharth Patwardhan
IBM Watson Research Center
1101 Kitchawan Road
Yorktown Heights NY 10598
siddharth@us.ibm.com

Branimir Boguraev
IBM Watson Research Center
1101 Kitchawan Road
Yorktown Heights NY 10598
bran@us.ibm.com

Apoorv Agarwal*
Dept. of Computer Science
Columbia University
New York, NY 10027
apoorv@cs.columbia.edu

Alessandro Moschitti
University of Trento
Via Sommarive 5
38123 Povo (TN), Italy
moschitti@disi.unitn.it

Jennifer Chu-Carroll
IBM Watson Research Center
1101 Kitchawan Road
Yorktown Heights NY 10598
jencc@us.ibm.com

ABSTRACT

State-of-the-art approaches to token labeling within text documents typically cast the problem either as a classification task, without using complex structural characteristics of the input, or as a sequential labeling task, carried out by a Conditional Random Field (CRF) classifier. Here we explore principled ways for structure to be brought to bear on the task. In line with recent trends in statistical learning of structured natural language input, we use a Support Vector Machine (SVM) classification framework deploying tree kernels. We then propose tree transformations and decorations, as a methodology for modeling complex linguistic phenomena in highly multi-dimensional feature spaces. We develop a general purpose tree engineering framework, which enables us to transcend the typically complex and laborious process of feature engineering. We build kernel-based classifiers for two token labeling tasks: fine-grained event recognition, and lexical answer type detection in questions. For both, we show that in comparison with a corresponding linear kernel SVM, our method of using tree kernels improves recognition, thanks to appropriately engineering tree structures for use by the tree kernel. We also observe significant improvements when comparing with a CRF-based realization of structured prediction, itself performing at levels comparable to state-of-the-art.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*text analysis, language parsing and understanding*

General Terms

Algorithms, Design

Keywords

support vector machines, tree kernels, token classification

*Work done at IBM Research during a summer internship.

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

1. INTRODUCTION

Among classification tasks in text analysis is a subclass characterized as assigning a binary label to each token in a given text span. Typically, the label assigned to any given token depends on the larger context in which the token is situated. Broadly speaking, current systems for such tasks are implemented as binary classification, either by suitably deriving features from the environment surrounding the token, or by using structured prediction inherent to sequence labeling, such as Conditional Random Field (CRF) [12].

One example of such a task is event recognition, where the system must identify words within a document that denote events—defined in their broad linguistic sense, as situations that happen or occur, denoted largely by verbs, event nominals, or predicative adjectives. Different approaches (discussed in more detail in the next section) appeal to some notions of structure (shallow parse, SVO trigrams, local dominance, and so forth), but these are not necessarily informed by principled analysis of feature informativeness.

For us, this offers an opportunity to explore advances in machine learning and classification, which actively uses the complex and intertwined characteristics of (linguistic) structure as complex features—in contrast to the approaches we discuss in the next section, approaches that do not use this structure in a principled way, and rely on manually engineered features.

Recent research [16] on statistical learning of natural language structured input seeks to develop methods for more direct encoding of inter-dependencies among elements of complex representations, both bypassing the need for non-trivial feature (and feature inter-dependence) engineering, and providing globally optimizing learning algorithms with more direct access to such data elements. Tree kernels, in particular, used within kernel methods (such as SVMs [26, 10, 27]), of-

fer an effective way to capture possible correlations between features and categories of features.

Consequently, many classification tasks where the data can be transformed into tree structures have been shown to benefit from tree kernels (see Section 2). Tree kernels are directly applied to such structures within the classification models, reducing, or even eliminating, the need for intensive feature definition based on introspection over domain knowledge. Feature engineering is thus replaced with the engineering of tree structures appropriate for the encapsulation of the essential information required for the effective application of tree kernels.

In general, the process of engineering trees is faster than engineering features, and therefore has the potential to become an enabling tool in the machine learning toolkit. However, existing approaches to engineering tree structures have been ad-hoc and largely task-specific. The literature provides little guidance on how one may go about generating tree structures for a class of tasks employing tree kernels.

In this paper, we begin to articulate a general methodology for generating tree representations appropriate to the general class of tasks outlined earlier—namely, those that require classifying tokens within their context. Our methodology provides a convenient way to rapidly engineer tree representations and to train quality classification models.

To develop the specific methodological steps, we defer to existing work using tree kernels, where a small number of "operations" are consistently used in engineering trees. These include, in particular, operations for tree transformations and enrichment (we offer an outline in Section 3 below), as primary components of a flexible mechanism for modeling complex linguistic phenomena in highly multi-dimensional feature space. It has been shown that such tree "pruning" and "decoration" operations improve system performance, as they can reduce structural variability and inject contextual and semantic information into the feature space. Consequently, we have developed a general-purpose tree engineering framework, which supports tree transformations, including pruning and decorating to facilitate rapid experimentation. We motivate our design with the need for exposing data elements of different nature to the classification algorithms.

The framework we describe builds on the basic notion of representing classifier instances as trees, with the operations above providing for enrichment and focusing of these trees. We also present the particular strategies for tree adaptations aimed at using structured input, given general characteristics of a category of classification tasks.

In this work, we also use our general methodology to investigate the novel application of tree kernels to two token labeling tasks. First, we demonstrate the effectiveness of our approach by showing results for event recognition, where our best performing system is a combination of linear and tree kernels, utilizing tree shapes derived from complete parses. In this setting, the system also stacks well against the accepted baseline for the task. We note that while SVMs have been previously used for event detection, these have not used tree kernels. In our second application, we then follow the same approach, but for a very different token classification task: lexical answer type detection. The task is to find within a given question, the word that expresses the type of the answer (a more detailed description of the task is presented in Section 4). This is a particularly critical component of open-domain question answering. Again, an SVM classifier capturing context with our tree kernel methodology, in addition to a manually engineered feature set, shows improvement over systems using only manually engineered features.

2. RELATED WORK

In this section, we first summarize existing work that employs tree kernels, and proffer that: (a) the techniques for generating tree representations are typically ad-hoc and task-specific, (b) have not been previously used to label tokens within their context. Second, we examine work in the literature addressing the two tasks to which we apply our methodology in this paper.

2.1 Tree Kernels and Kernel Engineering

Applying kernels, especially those for structured input, requires kernel engineering. This includes, in particular, the design of structures appropriate for the task at hand. Early work in this direction either used portions of syntactic constituency trees [19], or defined new dependency or shallow syntactic structures [28, 8]. More recent work promotes the notion of marking or enriching target nodes of constituency trees [29, 21], similarly to what we do (Section 3 below) for marking the target in our dependency structure.

The flexibility and effectiveness of tree kernels is apparent from the various types of tasks that they have been applied to—ranging from relation extraction [8] to semantic role labeling [21] to question classification [22]. In each of these, we find that the tree representations used by the tree kernels are derived from a syntactic parse or predicate argument structure (PAS), and typically transformed in some way for the task at hand. Based on observations of some common tree transformation operations in these previous approaches, we aim to establish a general purpose methodology in this paper. For instance, in the question classification work by Moschitti et al., we see that they use "slot" parent nodes in the PAS tree to indicate PAS argument labels. Our tree "decoration" operations described later in the paper are loosely based in this idea. Similarly, in the relation extraction work, Culotta and Sorensen keep only the smallest common subtree that includes the entities of the relation. This is akin to the tree pruning operation we describe later in the paper. Our main contribution in this paper is to provide a general purpose methodology to effectively apply tree kernels to a specific class of classification tasks and quickly enable the classifier to explore a rich feature space with little effort.

A related approach for exploring a richer feature space without explicit feature engineering is that of kernel engineering (e.g., [7]). However, this requires expertise in developing novel kernels for specific tasks. Our goal in this paper is to rely on existing tree kernel technology and focus our efforts, instead, on tuning the inputs to the kernel so as to achieve an appropriate feature space representation for our tasks. Our methodology provides the flexibility to add various types of information (such as semantic classes, named entities, etc.) within the same tree representation framework, enabling the feature space to capture complex interactions between syntactic, lexical and semantic attributes of each node in the tree. Such a feature space provides accurate and high generalization, which allows for learning classifiers with much less training data. Indeed, experiments in Section 5 confirm this for our tasks.

2.2 Event Detection

With small exceptions [25], all approaches to TimeBank-compliant[1] event recognition develop and train classifiers for token tagging. In most cases, the task is that of sequential labeling of tokens by encoding chunk information into token tags (IOB2 encoding scheme [24]). Classification, and interpretation of token sequences, differs across approaches: Boguraev and Ando [3] use a robust risk minimization classifier followed by Viterbi-style decoding; STEP [2] combines an SVM enhanced with a suite [11] for general-purpose chunking; Llorens et al. [15] use a CRF model.

Most approaches share intuitions about morpho-lexical properties as event indicators; they also share the view that extraneous information can be beneficial: Boguraev and Ando [3] use a word-profiling technique [1] to exploit large unannotated corpora for tagging/chunking, and Llorens et al. [15] attribute the performance boost they report to the use of semantic roles. Structural characteristics of the input, however, are used in these event recognition classifiers minimally, and are cumbersome in their implementation as features. For instance, some syntactic information is encoded in Boguraev and Ando's features as word uni- and bigrams based on subject-verb-object and preposition-noun constructions. Similarly, some phrasal information is rendered in feature form by Llorens et al., abstracted from a parse tree. Information (such as SRL and word profiling) beyond the 'base' feature set is clearly useful: for the two approaches cited, the 'base' classifier performance goes from 78.6%/78.67% (F-score) respectively, to 80.3%/81.40%.

There is no systematic way for such classifiers to capture extraneous information. As we will see (in Section 3) however, configurational and semantic information can be encoded—in a principled way—in a tree kernel by pruning and decorations. This is the primary line of investigation in our work.

2.3 Lexical Answer Type Detection

Question answering systems interchangeably use notions like 'focus', 'answer type' and 'lexical answer type' (LAT). Without going into finer details, these terms refer to the words that indicate the type of entity being asked for by the question. For example, in a question like *"He was a bank clerk in the Yukon before he published 'Songs of a Sourdough' in 1907."*, LATs are "he" and "clerk". LATs are different from semantic types, and detecting them is another example of context-dependent token classification task. A particularly thorough study of LAT detection is presented by Lally et al. [13]; salient to their work is the statistical LAT detection procedure presented, which sets the bar for state-of-the-art performance. Relevant intuitions leading to feature design can be found in previous work by Li and Roth [14]. In summary, a logistic regression classifier is trained for the task, with a feature set encapsulating numerous lexical and syntactic features, as well as the results of applying an extensive set of patterns developed specifically to match contexts indicative of LAT-ness. This classifier performs at close to 83.6% F-score.

3. TOKEN LABELING METHODOLOGY

As a classification task, event or LAT detection reduces to assigning a binary label to each token in a given document. Existing approaches include standard discriminative classifiers such as SVM, and sequence tagging approaches like CRF. All approaches follow a similar methodology. They analyze each token within a document in sequence, and identify a set of features likely to indicate the "eventness", or "LAT-ness" of that token. Machine Learning models are then trained using training data to classify each token as a positive or negative instance, based on the feature values associated with the token.

Many features capture only the context independent properties of the tokens — such as their part-of-speech, tense, lemma, etc. While these features are quite informative, they are not sufficient to achieve state-of-the-art performance. For instance, most events are represented as either verbs or nouns (typically nominalizations of verbs) and, as a result, part-of-speech is an important feature in the model. similar observations hold for the part-of-speech feature in a LAT model. However, beyond such inherent characteristics of tokens, their context is essential in resolving many of the more subtle or ambiguous cases. It is no surprise, then, that the best performing systems additionally model the context surrounding each target token.

The context surrounding the target token is typically captured by either including properties of neighboring tokens as features or by identifying certain structural clues surrounding the token, within the syntactic parse of the sentence. Take, for example, the event detection system implemented by Boguraev and Ando [3]. Their research employed features of the target token, such as: *token/capitalization/part-of-speech (POS) in 3-token window*, *bi-grams of adjacent words in 5-token window*, *words in the same syntactic chunk*, and *word unigrams/bigrams based on subject-verb-object* among others. Many such features were manually engineered to capture the context of a token through adjacency in a token window, through adjacency in syntactic chunks, and through proximity in the syntactic parse.

This type of feature engineering tends to rely heavily on human analysis of the data, and risks resulting in ad-hoc rules/patterns, that may not be able to model all the subtle pieces of contextual information required for the task. In this work, we take a more principled approach to modeling the context of tokens through the use of trees structures representing their context. SVM classifiers with tree kernels effectively exploit these tree structure to improve labeling performance over manually engineered features.

3.1 Tree Kernels

Tree kernels are an instance of a general class of convolution kernels. First introduced by Haussler [9], they can be used to compare abstract objects, like strings, instead of feature vectors. Convolution kernels involve a recursive calculation over the "parts" of abstract objects. This calculation is made computationally efficient by using *dynamic programming* techniques. By considering all possible combinations of fragments, tree kernels capture any possible correlation between features and categories of features.

Tree kernels in the past have been successfully applied for numerous Natural Language Processing tasks such as parser and tagger re-ranking [6], and question answer classification [22]. These are classification tasks where each example to be classified is a sentence (and, as such, has an associated tree). Cases where examples can be naturally represented

[1]TimeBank is distributed by the LDC; see LDC2006T08.

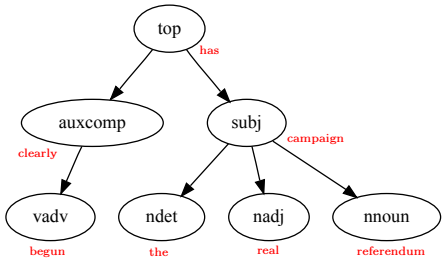

Figure 1: Dependency parse tree for the sentence: *the real referendum campaign has clearly begun.*

as trees also fall easily into the tree-kernel space (for e.g., semantic role labeling [21] and relation extraction [28, 8]).

More recently, the notion emerged that the tree representation in the kernel itself may be transformed—for instance, by pruning, to remove extraneous material [23], or by decorating, to add additional information [7]. Still not completely formalized, this notion is gaining ground; we will build upon it below.

In general, however, tree kernels have not been used for token labeling tasks like event or LAT detection. To the best of our knowledge, ours is the first work that explores the use of tree kernels, and presents a novel and effective data representation explicitly exploiting notions of tree transformations, for these tasks. We describe below the representation of the tree structures and motivation behind this design.

3.2 Tree Representation of a Token

While standard tree kernel classifiers have previously been used in numerous language processing tasks, the tree structures representing the instances to be classified in these tasks have typically been developed on a task-by-task basis. For many tasks, instances for classification can be naturally transformed into tree representations that are provided as input to tree kernels. For instance, in sentence classification, the parse trees of the sentences are tree structures that can be input for tree kernel classifiers. A token within a sentence, on the other hand, does not naturally lend itself to a tree representation. The key contribution of this paper is an approach for effectively capturing the context of a token with a tree representation so as to enhance the feature space of the classification task with standard tree kernel technology.

To generate the tree representation of a token within its context, we begin with the dependency parse of the sentence in which the target token appears. We map the token to a node within the parse tree, and then apply some tree transformations to generate the final contextual tree given as input to the classifier. Our tree transformations include node "decorating" operations and tree "pruning" operations around the token.

Before we begin decorating and transforming operations, we first obtain a tree that represents the basic structure of the sentence containing the target token. We take the dependency parse of a sentence and label each node with its syntactic role in the tree. Each node in the tree corresponds to a token from the sentence. Since the tree kernel ignores edge labels, these are dropped from the structure. This tree, with node labels representing the relationship of each node to its parent is the starting point for our tree operations to follow. Figure 1 is an example of this tree for the sentence: *the real referendum campaign has clearly begun.* Note that the labels in red, outside the nodes, are the tokens from the sentence corresponding to the respective nodes. They are for illustrative purposes only, and are not part of the data structure used in our tree operations.

Our tree operations then enable the classifier to essentially focus on the token to be classified, and to view its relationships with the immediate context of the token. The operations of focusing on a target token, and transforming the tree around it to both reduce noise and inject extra contextual information, are described in the following subsections.

3.2.1 *Identifying the Target*

Given a parse tree containing the target token to be classified, we would first like to be able to "identify" this token to the classifier in some way. Standard tree kernels operate on pairs of trees and compute a similarity score for each pair by counting the number of substructures that are common between the given input pair. The similarity score is essentially a function of these overlap statistics. Since we would like our trees to represent the relationships of the target token with its context, identifying the target token to the kernel essentially boils down to identifying any relationships between the target token and its neighboring nodes that are common between any two given trees. We try to achieve this by "decorating" or marking the target node in the tree. There are several possibilities that can be explored: (a) replace the label of the target node with an identifier (such as `-target-`) that is common across any pair of trees, (b) insert a parent node above the target, with an identifier that is common across trees, or (c) insert a child node below the parent, with an identifier that is common across trees.

In our work, we mark the target node by inserting a parent node with the label `-target-` above the target node. We tried several experiments, and found this to be the most effective for our tasks (the differences between the three methods, however, were very small). This decoration enables us to match relationships across trees between the target node and nodes above it in tree structures being compared by the kernel. Relationships between the target and nodes below it are matched by the kernel only if the target plays the same syntactic role in the trees being compared. Figure 2(a) illustrates the "target" node decoration for the example tree, with the node corresponding to the word "*campaign*" marked as the target.

3.2.2 *Pruning Around the Context*

Many of the trees we typically deal with can come from long sentences, and are complex trees with many nodes and edges. For our classification tasks, many of the nodes in the tree can be in a parts of the tree completely separate and unrelated to the target node. Such nodes should not be considered as a part of the local context of the target node, and should have little or no impact on the classification decision. Leaving such nodes in the tree can allow these to match within the kernel, and could artificially increase the kernel similarity score of a pair of trees that do not actually represent a similar context for the target tokens. In other words, large trees can introduce noise into the contextual representation of target tokens. We, therefore, prune the trees around the target token to eliminate such noise. As an

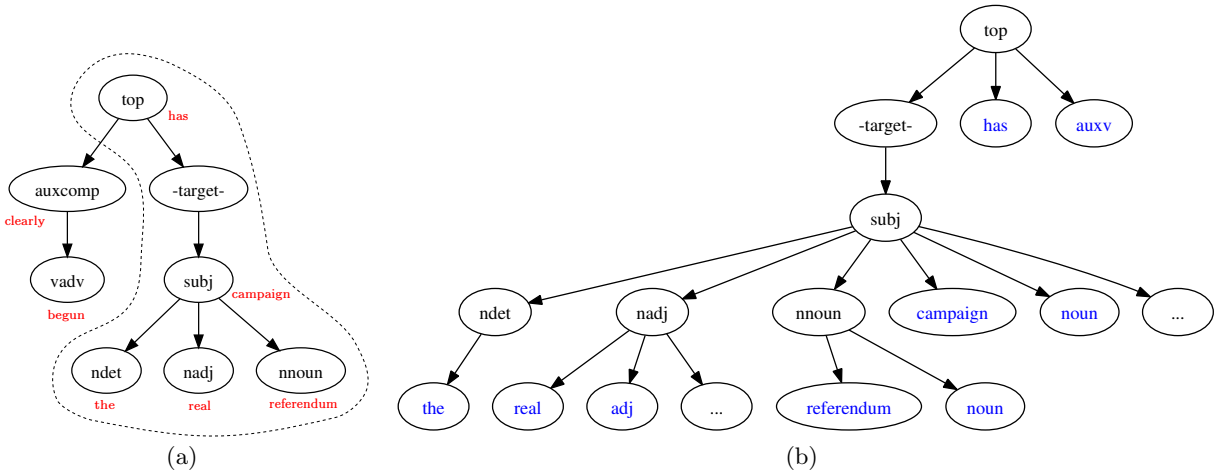

Figure 2: (a) Dependency parse tree for the sentence *the real referendum campaign has clearly begun.* The dotted line encloses the pruned tree. (b) Pruned tree with "decoration." Nodes in blue are subset of features we add to the tree nodes.

added benefit, we find that pruning also substantially speeds up the classifier (as it now needs to match fewer subtrees).

There are many possible ways in which a tree may be pruned to capture the essential context required by the classifier. For instance, one may choose to keep only the direct ancestors of the target as its required context. Alternatively, one may choose to base pruning decisions using a more principled linguistic approach using the types of parse nodes and edges for pruning. Depending on the task, this could mean keeping only the pre-modifiers and post-modifiers of the target or restricting the pruning to the clause in which the target appears. We found that for our tasks, an effective pruning strategy was to keep only the direct ancestors and child nodes of the target, and discard all other nodes and edges in the tree. The dotted line in Figure 2(a) illustrates this pruning approach for the example sentence.

3.2.3 *Decorating the Nodes*

Having pruned the tree to restrict the context around the target, and marked the target node within the tree, we now perform one additional (and important) step to better represent the context of the target — node decoration. Observe that after pruning, the tree contains only the information about the syntactic relationships of each node with its neighbors. Hidden in each node is a wealth of additional information about the context that could be effectively exposed in the tree. For instance, each node in the tree represents a token, which has a lexical surface form, part of speech, semantic class information, etc. that could be effectively used by the classifier in its decisions. We incorporate these "node features" as "decorations" on the tree.

We decorate each node corresponding to a token with token features by inserting them as new child nodes with labels encoding the token features. For instance, we add new child nodes "lemma=campaign", "part-of-speech=noun", etc. to the node corresponding to the token "campaign". Figure 2(b) illustrates these new token feature nodes (with blue labels) in our example tree. Due to space constraints we do not show the feature names ("lemma=", "part-of-speech=", etc.) in the figure. The set of node features can be obtained from existing NLP tools, such as the parser, a named entity tagger, semantic class detector, and the like.

This tree representation combines many categories of features (lexical, syntactic and semantic) in one succinct representation. This gives the learner an opportunity to learn combinations of features which would otherwise be tedious, if not impossible, for humans to encode in an explicit feature space. For this reason, we use a Partial Tree (PT) kernel (first proposed by Moschitti [20]) for defining the implicit feature space of the learner. A PT kernel calculates the similarity between two trees by comparing all possible subtrees. For example, some of the subtrees in the implicit feature space of a partial tree kernel on the tree in Figure 2(b) are shown in Figure 3. This means that the tree kernel attempts to combine all of the information available to it in different ways, resulting in various levels of abstraction depending on which pieces of information were selected for a particular subtree. One subtree may drop all lexical items of the selected nodes, keeping only syntactic and semantic information for the included nodes. Another may keep only the lexical information. And yet another have a combination of these different pieces of information for different nodes in the subtree. As one would imagine, this results in a massive (implicit) feature space of all possible subtrees. However, learning and classification is made tractable by the kernel, which uses dynamic programming for an efficient enumeration of this space. Thus, with the use of this approach, it is not necessary to manually engineer features at all levels of abstraction.

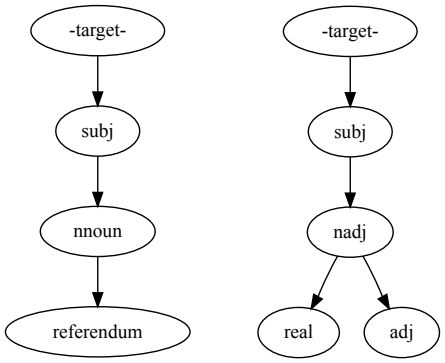

Figure 3: Example partial trees in tree kernel feature space.

4. TASK DEFINITION AND DATA SET

The event detection task is a token-in-context labeling problem, in the sense defined earlier, in Section 1. Conventionally, however, many of the approaches to date (as discussed in Section 2) treat it as a sequential tagging problem. For us, it is an instance of a more general category of classification problems, interesting because of its intrinsic characteristics which make it a suitable example for the application of our tree kernel-based methods. In order to demonstrate broader applicability of these methods, we also focus on another token labeling task, namely that of LAT detection. The results we report below (Section 5) underpin our claim for the effectiveness of our tree transformations-based methods of leveraging tree kernels for token labeling. In future work, we plan to apply these methods to other token labeling tasks, such as, for instance, markable detection for coreference resolution.

4.1 Event Recognition

Events in general text may be expressed in a variety of ways; morpho-syntactically, they may be realized by means of untensed or tensed verbs, nominalizations, predicative adjectives or clauses, or certain prepositional phrases. For example, only some of the nouns and verbs in the following sentence are to be labeled as *event* mentions.

Several pro-Iraq demonstrations have taken place in the last week.

In line with the event recognition work discussed in Section 2, we use the TimeBank event-specific data set to train our models, as well as to evaluate their performance on the event recognition task.

The distribution of event mentions in the corpus follows the natural distribution of event-denoting linguistic tokens in (news) discourse. There are 56,632 (non-punctuation) tokens in TimeBank, of which 7,921 are captured by event annotations. TimeBank is relatively small, and Boguraev and Ando [4] discuss its make-up in some detail.

Practically, only a tiny proportion of event instances in the data are realized as multi-token mentions. In fact, the guidelines for TimeBank-compliant event annotation are heavily leaning to a convention which favors marking single tokens as 'proxies' for event-denoting phrases. It turns out that in the entire TimeBank corpus (∼57K word tokens), there are only 30 or so multi-token event mentions. This imbalance is such that the rationale for casting the task as that of multi-class sequential tagging—as most of the approaches in Section 2 do—is questionable: it is unreasonable to expect that concurrent models can be built for 'begin' and 'inside' event tags from such impoverished data.

From our point of view, this is sufficient justification to model the event recognition task as a binary token classification.

4.2 Lexical Answer Type Detection

The Lexical Answer Type (LAT) detection task (a subtask of question answering) is also a binary token classification problem of identifying an answer type string in a question. For example, in a question like:

Which US president was first to be re-elected for a second term?

the text span *"president"* is the LAT (it identifies what type of entity is being asked for in the question). Multi-token LATs may apparently (and occasionally) be encountered in the data (e.g. *"vice president"*), but our parser (see Section 5.1 below) handles these as single tokens.

Feature	Description
pos	The part of speech of the token
surface	The lexical surface form of the token
lemma	The morphological root of the token
semantic	Semantic class features based on a named-entity recognition system
parse	Parse features from a set of 44 syntactic properties assigned by the parser

Table 1: The set of features we add as children to the nodes of the dependency parse tree.

The dataset we use is the one described in Lally, et al. [13]. It is a manually annotated set of 9,128 Jeopardy! questions. In terms of class distribution, this question data set contains of the order of 111,597 tokens, 11,906 of which are positive (LAT) examples. The dataset is split into 8,000 training and 1,128 test questions (97,741 training tokens and 13,856 testing). Given the observation in Section 2, concerning the relatively small volumes of training data sufficient for learning good classification functions when the tree kernel can capture dependencies between lexical, syntactic and semantic features, this size of dataset turns out to be adequate. We will return to this point in the next section (5).

5. EVALUATION AND DISCUSSION

Within the framework and methodology described in the previous section, we have developed classifiers for event detection and LAT detection. In both cases, we demonstrate the effectiveness of tree kernels, by enhancing an existing manually feature engineered model with our tree kernel. This is done by implementing the manually feature engineered model using a linear kernel, and combining that with our tree kernel. Note that by design, in the tree kernels settings there are additional features in the feature space: this reflects our approach of letting the kernel identify salient structural characteristics present in the data, and use them as features in a highly multi-dimensional space. The combination of linear and tree kernels is done using a composite kernel that sums the two. Our SVM classifier uses the composite kernel for the task at hand.

5.1 Implementation

In this work, we use dependency parses derived from the output of the English Slot Grammar parser [17]. Each token in that sentence then maps to a specific node in the dependency parse. A node is only labeled with the syntactic role of the token within this parse. We then apply the tree transformations (described in Section 3) to obtain the "contextual trees" for the tokens. This includes marking the target with a parent node, pruning the tree around the target, and inserting feature "decorations" as child nodes. The various types of information that we attach as child nodes are obtained from the syntactic parser are listed in Table 1.

5.2 Experimental Setup

Our experiments are designed to investigate our claims: namely, that combining linear and tree kernels outperforms individual kernel settings, and that a tree kernel utilizing pruning and decorations in a particular way can effectively

drive a classifier (SVM) for token labeling tasks. For both tasks, a model based on the linear kernel uses a manually engineered set of features. For event detection, this is loosely based on the feature set from Boguraev and Ando [3]. For LAT detection, we refer to the feature set from Lally et al. [13]. The features broadly fall into lexical, morphological, and syntactic categories, and capture sequentiality of tokens by observing occurrences of lexical elements (tokens, lemmas), and/or their properties (orthography, part-of-speech, inflectional information) in 3- or 5-token windows.

The features include: token, capitalization, part-of-speech (POS) in 3-token window; bigrams of adjacent words in 5-token window; and trigrams of POS, capitalization and word suffix. To the extent that it is used, structural information is derived largely by observing the output of a parser. In contrast to the work by Boguraev and Ando [3], who use a shallow parser, we extract features from a deep syntactic tree; this allows us to have reliable word uni- and bi-grams based on subject-verb-object and preposition-noun constructions. Additional grammatical information is injected into the feature space by mapping the labels of immediately dominating phrases, and by windowing over adjacent base phrasal units: we derive features for words in the same (base) syntactic phrase, and for head-words in 3-phrase window.

The two sets of features (for event, and for LAT, detection) largely overlap; the differences, where they exist, stem from the different nature of the two tasks. For instance, for lexical answer types, the shape of the encompassing phrasal unit (typically a noun phrase) is very important; consequently, there are a large number of structural pattern rules, which detect LAT-indicative contexts. Even so, as the results show, the tree kernels are able to identify additional salient features.

Further, and in contrast to Boguraev and Ando [3], who do not use semantic features, we bring semantics into the feature space by deferring to the semantic properties of parse nodes, as they are constructed by the parser and integrated into larger, semantically coherent, constituents. Our parser implements its own set of ontological types, and some of these bear directly upon both the task of determining the eventness of a verb or a noun and the task of determining that a word in a question acts as its lexical answer type.

All the feature sets from Boguraev and Ando [3], Lally et al. [13], and our implementations have been shown to achieve good accuracy at their respective tasks. Overall, the features have been manually engineered to represent the attributes of the token and its surrounding context within the linear kernel. An SVM trained only on the linear kernel forms a baseline system for comparison.

Given that the tasks are somewhat akin to sequence labeling, our experiments also include results from applying a linear chain CRF [12], a popular and effective discriminative technique for supervised sequence labeling. CRFs are undirected graphical models, a special case of conditionally-trained finite state machines. This makes them appropriate (especially in the event detection case), since a CRF by its nature does a form of structured prediction, and thus incorporates the structure (albeit only "sequential" structure) of the data in its decision making process.

Intended as an additional baseline, our CRF implementations[2] [5] use the same feature sets used by the corresponding linear kernels (see below), for both tasks. In the case of event detection, our CRF baseline runs at a level comparable to the state-of-the-art.[3] In the case of LAT detection, there are no CRF results reported to compare to, but our CRF-based LAT classifier offers a data point which further confirms the superiority of the framework and method we propose here.

[2]We use the CRF implemented in the MinorThird package; see `http://sourceforge.net/apps/trac/minorthird/`

	Acc	P	R	F
	Event Detection			
SVM-lin	94.16%	0.753	0.853	0.800
CRF	93.86%	0.765	0.811	0.787
SVM-prune-tree	92.90%	0.709	0.818	0.760
SVM-full-tree+lin	94.04%	0.743	0.864	0.799
SVM-prune-tree+lin	94.36%	0.754	0.874	0.810
	LAT Detection			
SVM-lin	96.26%	0.806	0.853	0.829
CRF	96.43%	0.870	0.782	0.823
SVM-prune-tree	94.33%	0.719	0.765	0.741
SVM-full-tree+lin	-	-	-	-
SVM-prune-tree+lin	96.46%	0.816	0.861	0.838

Table 2: Experimental results.

5.3 Results

Table 2 presents the results of our experiments. All our event detection models are trained on a 150 document training set of randomly chosen documents from the 183 Timebank documents. We evaluate these on the remaining 33 documents. Similarly, out LAT detection models are trained on 8,000 Jeopardy! questions and evaluated on 1,128 questions, all annotated with LATs. We report overall accuracy and also precision, recall and F-score on the positive class. The row labeled SVM-lin contains the performance of the linear kernel SVM using the manually engineered feature set; the row labeled CRF contains the performance of our CRF baseline; the row labeled SVM-prune-tree contains the performance of our tree kernel SVM model; the row labeled SVM-full-tree+lin contains the performance of our model which is a combination of the linear and full parse tree kernel; the row labeled SVM-prune-tree+lin contains the performance of our model which is a combination of the linear and pruned parse tree kernel. We were unable to generate performance scores for LAT detection with the SVM-full-tree+lin configuration primarily due to the sheer size of the trees and the size of the data set, that required for too many computational resources for the experiment to complete successfully.

Our results show that our SVM-prune-tree+lin configuration outperforms the CRF baseline—for both tasks—by 2.3% F-score absolute on event detection, and by 1.5% F-score absolute on LAT detection. As we pointed out earlier (Section 5.2), our CRF classifier for event detection is running at state-of-the-art level, as its feature set compares with an experimental setup (orthographic, morpho-syntactic, and lexical semantic, without semantic roles) reported by Llorens et al. [15]: F=78.67%. The additional claim of that work, that CRFs outperform SVMs for event detection, refers to a very different SVM configuration to ours, and should only be considered in the context of that work.

[3]See Section 2, where we highlight the accuracy levels of the 'base' classifiers used by Boguraev and Ando [3] and Llorens et al. [15], which are very close, at 78.6% and 78.67% respectively.

	Acc	P	R	F
	Event Detection			
SVM-prune-tree+lin	94.36%	0.754	0.874	0.810
ablate mark-target	94.29%	0.752	0.871	0.807
ablate decorations	94.10%	0.742	0.874	0.802
	LAT Detection			
SVM-prune-tree+lin	96.46%	0.816	0.861	0.838
ablate mark-target	96.42%	0.812	0.862	0.836
ablate decorations	96.24%	0.804	0.854	0.828

Table 3: Ablating mark target and tree decorations.

Our best system also outperforms the SVM-lin baseline by 1.0% F-score absolute (statistically significant with $p < 0.05$ using McNemar's test [18]) for event detection and by 0.9% F-score absolute (statistically significant with $p < 0.06$) on LAT detection. While tree kernels alone perform worse than the two baselines, when combined with the linear feature space, we achieve our best performing system. Clearly, there is structural information offered by the kernel design that is absent in both the CRF and linear kernel baseline.

Our results show that pruning is crucial. In comparison to the configuration using the full parse tree (SVM-full-tree+lin), the pruned version (SVM-prune-tree+lin) performs substantially better. We also measured the impact of marking the target nodes in the trees and that of decorating the nodes with features as child nodes by ablating these from the SVM-prune-tree+lin configuration. Table 3 presents these results. The rows labeled "ablate mark-target" are the results for the SVM-prune-tree+lin configuration without the target node marked with parent node. The rows labeled "ablate mark-target" are the results for the SVM-prune-tree+lin configuration without the child node decorations. In both tasks, we see that the feature decorations clearly have the most impact, while marking the target node has a lower (but positive) impact.

Previous work has also shown that the use of tree kernels can reduce the amount of training data required to train SVM classifiers. We demonstrate that this is also true for our tasks by generating learning curves. We evaluate models trained on randomly sampled subsets of the training data. Figures 4 and 5 present the learning curves for the event detection and LAT detection, respectively. The x-axis in each graph is the percentage of training data used for training model, and the y-axis represents the F-score of the model on the test data. Each figure has one graph (Linear) generated for the basline (SVM-lin) configuration of the SVM and one graph (Linear+TK) for the tree kernel (SVM-prune-tree+lin) configuration. It is clear from both figures that the tree kernels can learn a good model with far fewer training examples as compared to the linear kernel baseline.

In our experiments, we do not offer direct comparison with related work. This reflects, partly, lack of a uniform baseline. While performance is reported in experimental settings with cross-validation over the entire corpus, the nature and relative weight of the individual folds are not identical. As there is no develop/train/test split of the reference corpus (stratified or otherwise), the measures reported in the literature can not be directly comparable. In any case, our primary focus in this work is not to come up with an event, or LAT detection device, which will best, say, CRF methods for the same task; rather, we are interested in the novel use of structured input (tree-kernel-based) for a sequence labeling task, which allows us to implement an alternative classification framework, performing at, or close to, state-of-the-art levels. Even though we cannot directly compare our systems with the state-of-the-art, our performance numbers suggest that both our systems are within state-of-the-art levels.

6. CONCLUSIONS

The work presented in this paper was originally framed as a classification-based framework for recognizing events in text. Fundamentally, however, we were motivated to study the applicability, and effectiveness, of recent work on tree kernel-based methods for structured-input based learning. More specifically, the event detection task highlighted a class of labeling applications where the goal is to assign binary labels to token targets, and where correct labeling crucially depends on the larger context.

Common to the classification frameworks developed for token labeling tasks are the intrinsic characteristics of feature sets for the respective classifiers: manual feature engineering underpins all efforts, and invariably is largely concerned with exposing lexical, morpho-syntactic, and in some cases semantic features. While it is recognized that contextual dependencies are of importance, corresponding features capture only approximations of such dependencies, within a tiny window, at that.

Structural characteristics of the inputs, which may well carry significant signal for the task at hand, are hardly encoded in the feature set. We propose a set of extensions to an SVM-based classifier deploying tree kernels, such that the information intrinsic in a parse tree representation of the input can be directly folded into multi-dimensional feature space. 'Raw' trees, however, may be somewhat distracting, as well as incomplete with respect to the broad set of data categories which may be of relevance to the classification task. Our framework builds on notions of tree pruning and decorating, to provide for refocusing the learning algorithms to appropriate signal-bearing tree fragments, additionally adorning them with extrinsic information. In effect, we re-introduce feature engineering, but instead of introspecting about individual features, we allow for such multiple features, and inter-dependencies therebetween, to be directly accessible by globally optimizing learning algorithms.

In the methodology we have developed, we recast the token labeling tasks appropriately, re-purposing traditional features as decorations on suitably pruned tree fragments; these now become richly adorned representations of the token targets, and the focus of new classifiers deploying tree kernels, which capture structural inter-dependencies between contextually related data elements. We have instantiated this methodology for two diverse labeling tasks, event recognition and lexical answer type detection. Our results show that the information coming from the tree kernel complements the information captured in a manually engineered set of features. This is manifested in better resuts: the new (composite kernel) classifiers show statistically significant improvement over configurations using manually engineered features.

7. ACKNOWLEDGEMENTS

This research is supported in part by Air Force Contract FA8750-09-C-0172 under the DARPA Machine Reading Program.

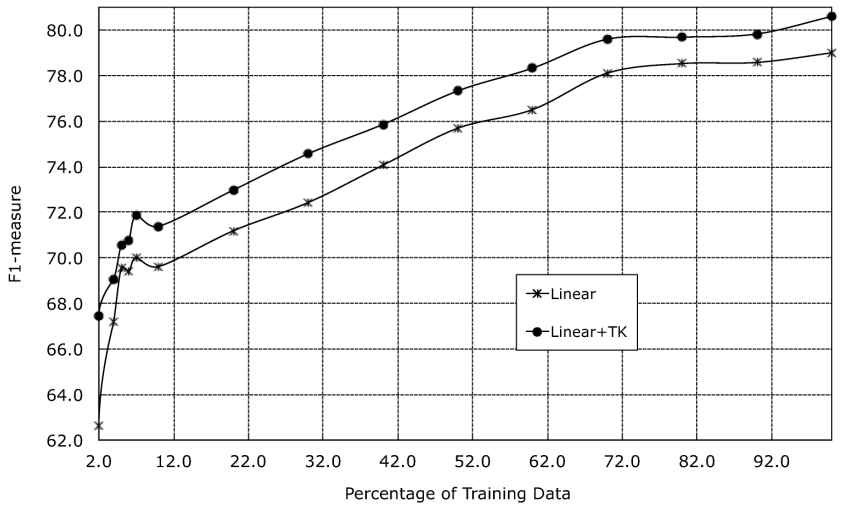

Figure 4: Learning curve for event detection.

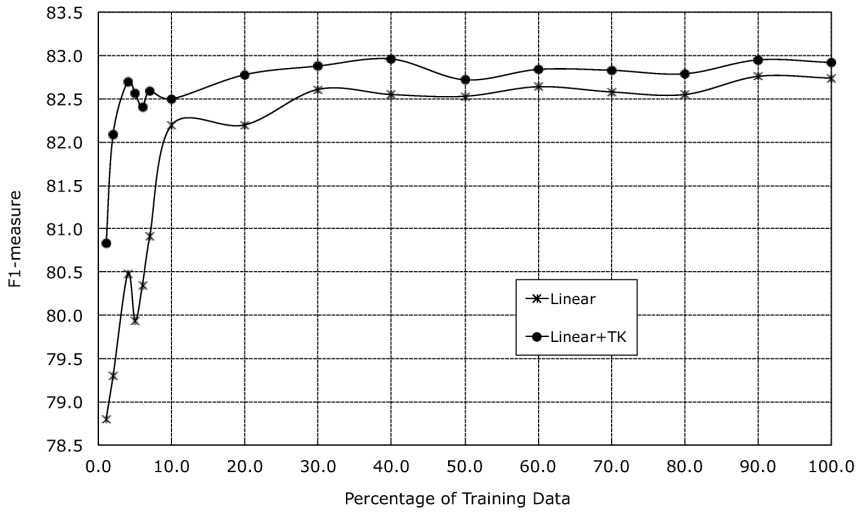

Figure 5: Learning curve for LAT detection.

8. REFERENCES

[1] R. K. Ando. Exploiting Unannotated Corpora for Tagging and Chunking. In *The Companion Volume to the Proceedings of 42st Annual Meeting of the Association for Computational Linguistics*, pages 142–145, Barcelona, Spain, July 2004.

[2] S. Bethard and J. H. Martin. Identification of Event Mentions and their Semantic Class. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 146–154, Sydney, Australia, July 2006.

[3] B. Boguraev and R. K. Ando. TimeML-compliant Text Analysis for Temporal Reasoning. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 997–1003, Edinburgh, Scotland, August 2005.

[4] B. Boguraev and R. K. Ando. Analysis of TimeBank as a Resource for TimeML Parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, pages 71–76, Genoa, Italy, May 2006.

[5] W. Cohen. MinorThird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data. `http://minorthird.sourceforge.net`, 2004.

[6] M. Collins and N. Duffy. Convolution kernels for Natural Language. In *Advances in Neural Information Processing Systems*, Vancouver, Canada, December 2001.

[7] D. Croce, A. Moschitti, and R. Basili. Structured Lexical Similarity via Convolution Kernels on Dependency Trees. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1034–1046, Edinburgh, Scotland, July 2011.

[8] A. Culotta and J. Sorensen. Dependency Tree Kernels

for Relation Extraction. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04)*, pages 423–429, Barcelona, Spain, July 2004.

[9] D. Haussler. Convolution Kernels on Discrete Structures. Technical Report UCSC-CRL-99-10, University of California at Santa Cruz, July 1999.

[10] V. Kecman. *Learning and Soft Computing*. The MIT Press, Cambridge, MA, 2001.

[11] T. Kudo and Y. Matsumoto. Chunking with Support Vector Machines. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 192–199, Pittsburgh, PA, June 2001.

[12] J. Lafferty, A. McCallum, and F. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289, June 2001.

[13] A. Lally, J. Prager, M. McCord, B. Boguraev, S. Patwardhan, J. Fan, P. Fodor, and J. Chu-Carroll. Question Analysis: How Watson Reads a Clue. *IBM Journal of Research and Development*, 56(3/4):2:1–2:14, May/July 2012.

[14] X. Li and D. Roth. Learning Question Classifiers: The Role of Semantic Information. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 556–562, Taipei, Taiwan, August 2004.

[15] H. Llorens, E. Saquete, and B. Navarro-Colorado. TimeML Events Recognition and Classification: Learning CRF Models with Semantic Roles. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 725–733, Beijing, China, August 2010.

[16] L. Màrquez and A. Moschitti. Special Issue on Statistical Learning of Natural Language Structured Input and Output. *Natural Language Engineering*, 18(2):147–153, April 2012.

[17] M. McCord. Slot Grammar: A System for Simpler Construction of Practical Natural Language Grammars. In *Proceedings of the International Symposium on Natural Language and Logic*, pages 118–145, Hamburg, Germany, May 1989.

[18] Q. McNemar. Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages. *Psychometrika*, 12(2):153–157, 1947.

[19] A. Moschitti. A Study on Convolution Kernels for Shallow Statistic Parsing. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04)*, pages 335–342, Barcelona, Spain, July 2004.

[20] A. Moschitti. Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *Proceedings of the 17th European Conference on Machine Learning*, pages 318–329, Berlin, Germany, September 2006.

[21] A. Moschitti, D. Pighin, and R. Basili. Tree Kernels for Semantic Role Labeling. *Computational Linguistics*, 34(2):193–224, June 2008. Special Issue on Semantic Role Labeling.

[22] A. Moschitti, S. Quarteroni, R. Basili, and S. Manandhar. Exploiting Syntactic and Shallow Semantic Kernels for Question Answer Classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 776–783, Prague, Czech Republic, June 2007.

[23] T.-V. T. Nguyen, A. Moschitti, and G. Riccardi. Convolution Kernels on Constituent, Dependency and Sequential Structures for Relation Extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1378–1387, Singapore, August 2009.

[24] L. Ramshaw and M. Marcus. Text Chunking using Transformation-Based Learning. In *Proceedings of Third Annual Workshop on Very Large Corpora*, pages 82–94, Cambridge, MA, June 1995.

[25] R. Saurí, R. Knippen, M. Verhagen, and J. Pustejovsky. Evita: A Robust Event Recognizer For QA Systems. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 700–707, Vancouver, Canada, October 2005.

[26] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, NY, 1995.

[27] L. Wang, editor. *Support Vector Machines: Theory and Applications*. Springer, Berlin, Germany, 2005.

[28] D. Zelenko, C. Aone, and A. Richardella. Kernel Methods for Relation Extraction. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 71–78, Philadelphia, PA, July 2002.

[29] M. Zhang, J. Zhang, and J. Su. Exploring Syntactic Features for Relation Extraction using a Convolution Tree Kernel. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 288–295, New York City, USA, June 2006.