Dependency Tree Kernels for Relation Extraction

Aron Culotta
University of Massachusetts
Amherst, MA 01002
culotta@cs.umass.edu

Jeffrey Sorensen
IBM T.J. Watson Research Center
Yorktown Heights, NY
sorenj@us.ibm.com

Abstract

We extend previous work on tree kernels to estimate the similarity between the dependency trees of sentences. Using this kernel within a Support Vector Machine, we detect and classify relations between entities in the Automatic Content Extraction (ACE) corpus of news articles. We examine the utility of different features such as Wordnet hypernyms, parts of speech, and entity types, and find that the dependency tree kernel achieves a 20% F1 improvement over a "bag-of-words" kernel.

1 Introduction

The ability to detect complex patterns in data is limited by the complexity of the data's representation. In the case of text, a more structured data source (e.g. a relational database) allows richer queries than does an unstructured data source (e.g. a collection of news articles). For example, current web search engines would not perform well on the query, "list all California-based CEOs who have social ties with a United States Senator." Only a structured representation of the data can effectively provide such a list.

The goal of Information Extraction (IE) is to discover relevant segments of information in a data stream that will be useful for structuring the data. In the case of text, this usually amounts to finding mentions of interesting entities and the relations that join them, transforming a large corpus of unstructured text into a relational database with entries such as those in Table 1.

Entity	Type	Location
Apple	Organization	Cupertino, CA
Microsoft	Organization	Redmond, WA

Table 1: An example of extracted fields

IE is commonly viewed as a three stage process: first, an *entity tagger* detects all mentions of interest; second, *coreference resolution* resolves disparate mentions of the same entity; third, a *relation extractor* finds relations between these entities. Entity tagging has been thoroughly addressed by many statistical machine learning techniques, obtaining greater than 90% F1 on many datasets (Tjong Kim Sang and De Meulder, 2003). Coreference resolution is an active area of research not investigated here (Pasula et al., 2002; McCallum and Wellner, 2003).

We describe a relation extraction technique based on *kernel methods*. Kernel methods are non-parametric density estimation techniques that compute a *kernel function* between data instances, where a kernel function can be thought of as a similarity measure. Given a set of labeled instances, kernel methods determine the label of a novel instance by comparing it to the labeled training instances using this kernel function. Nearest neighbor classification and support-vector machines (SVMs) are two popular examples of kernel methods (Fukunaga, 1990; Cortes and Vapnik, 1995).

An advantage of kernel methods is that they

AT	NEAR	PART	ROLE	SOCIAL
Based-In Located Residence	Relative-location	Part-of Subsidiary Other	Affi liate, Founder Citizen-of, Management Client, Member Owner, Other, Staff	Associate, Grandparent Parent, Sibling Spouse, Other-professional Other-relative, Other-personal

Table 2: Relation types and subtypes.

can search a feature space much larger than could be represented by a feature extraction-based approach. This is possible because the kernel function can explore an *implicit* feature space when calculating the similarity between two instances, as described in the Section 3.

Working in such a large feature space can lead to over-fitting in many machine learning algorithms. To address this problem, we apply SVMs to the task of relation exraction. SVMs find a boundary between instances of different classes such that the distance between the boundary and the nearest instances is maximized. This characteristic, in addition to empirical vaidation, indicates that SVMs are particularly robust to over-fitting.

Here we are interested in detecting and classifying instances of relations, where a relation is some meaningful connection between two entities (Table 2). We represent each relation instance as an *augmented dependecy tree*. A dependency tree represents the grammatical dependencies in a sentence; we augment this tree with features for each node (e.g. part of speech) We choose this representation because we hypothesize that instances containing similar relations will share similar substructures in their dependency trees. The task of the kernel function is to find these similarities.

We define a *tree kernel* over dependency trees and incorporate this kernel within an SVM to extract relations from newswire documents. The tree kernel approach consistently outperforms the bag-of-words kernel, suggesting that this highly-structured representation of sentences is more informative for detecting and distinguishing relations.

2 Related Work

Kernel methods (Vapnik, 1998; Cristianini and Shawe-Taylor, 2000) have become increasingly popular because of their ability to map arbitrary objects to a Euclidian feature space. Haussler (1999) describes a framework for calculating kernels over discrete structures such as strings and trees. String kernels for text classification are explored in Lodhi et al. (2000), and tree kernel variants are described in (Zelenko et al., 2003; Collins and Duffy, 2002; Cumby and Roth, 2003). Our algorithm is similar to that described by Zelenko et al. (2003). Our contributions are a richer sentence representation, a more general framework to allow feature weighting, as well as the use of composite kernels to reduce kernel sparsity.

Brin (1998) and Agichtein and Gravano (2000) apply pattern matching and wrapper techniques for relation extraction, but these approaches do not scale well to fastly evolving corpora. Miller et al. (2000) propose an integrated statistical parsing technique that augments parse trees with semantic labels denoting entity and relation types. Whereas Miller et al. (2000) use a generative model to produce parse information as well as relation information, we hypothesize that a technique discriminatively trained to classify relations will achieve better performance. Also, Roth and Yih (2002) learn a Bayesian network to tag entities and their relations simultaneously. We experiment with a more challenging set of relation types and a larger corpus.

3 Kernel Methods

In traditional machine learning, we are provided a set of training instances $S = \{x_1 \ldots x_N\}$, where each instance x_i is represented by some d-dimensional feature vector. Much time is spent on the task of *feature engineering* – searching for the optimal feature set either manually by consulting domain experts or automatically through feature induction and selection (Scott and Matwin, 1999). For example, in entity detection the original instance representation is generally a word vector corresponding to a sentence. Feature extraction and induction may result in features such as part-of-speech, word n-grams, character n-grams, capitalization, and conjunctions of these features. In the case of more structured objects, such as parse trees, features may include some description of the object's structure, such as "has an NP-VP subtree." Kernel methods can be particularly effective at reducing the feature engineering burden for structured objects. By calculating the similarity between two objects, kernel methods can employ dynamic programming solutions to efficiently enumerate over substructures that would be too costly to explicitly include as features.

Formally, a kernel function K is a mapping $K : \mathbf{X} \times \mathbf{X} \to [0, \infty]$ from instance space $\mathbf{X}$ to a similarity score $K(x, y) = \sum_i \phi_i(x)\phi_i(y) = \phi(x) \cdot \phi(y)$. Here, $\phi_i(x)$ is some feature function over the instance x. The kernel function must be *symmetric* $[K(x, y) = K(y, x)]$ and *positive-semidefinite*. By positive-semidefinite, we require that the if $x_1, \ldots, x_n \in \mathbf{X}$, then the $n \times n$ matrix G defined by $G_{ij} = K(x_i, x_j)$ is positive semi-definite. It has been shown that any function that takes the dot product of feature vectors is a kernel function (Haussler, 1999).

A simple kernel function takes the dot product of the vector representation of instances being compared. For example, in document classification, each document can be represented by a binary vector, where each element corresponds to the presence or absence of a particular word in that document. Here, $\phi_i(x) = 1$ if word i occurs in document x. Thus, the kernel function $K(x, y)$ returns the number of words in common between x and y. We refer to this kernel as the "bag-of-words" kernel, since it ignores word order.

When instances are more structured, as in the case of dependency trees, more complex kernels become necessary. Haussler (1999) describes *convolution kernels*, which find the similarity between two structures by summing the similarity of their substructures. As an example, consider a kernel over strings. To determine the similarity between two strings, string kernels (Lodhi et al., 2000) count the number of common subsequences in the two strings, and weight these matches by their length. Thus, $\phi_i(x)$ is the number of times string x contains the subsequence referenced by i. These matches can be found efficiently through a dynamic program, allowing string kernels examine long-range features that would be computationally infeasible in a feature-based method.

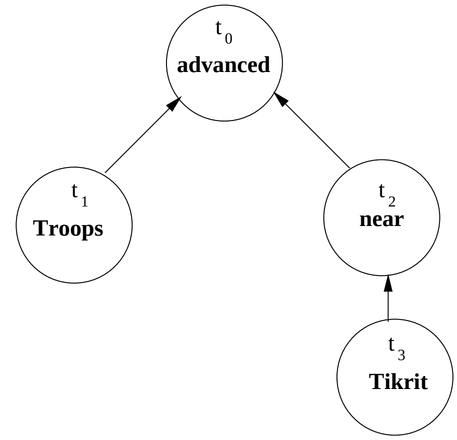

Figure 1: A dependency tree for the sentence *Toops advanced near Tikrit.*

Given a training set $S = \{x_1 \ldots x_N\}$, kernel methods compute the *Gram* matrix G such that $G_{ij} = K(x_i, x_j)$. Given G, the classifier finds a hyperplane which separates instances of different classes. To classify an unseen instance x, the classifier first projects x into the feature space defined by the kernel function. Classification then consists of determining on which side of the separating hyperplane x lies.

A *support vector machine* (SVM) is a type of classifier that formulates the task of finding the separating hyperplane as the solution to a quadratic programming problem (Cristianini and Shawe-Taylor, 2000). Support vector machines attempt to find a hyperplane that not only separates the classes but also maximizes the margin between them. The hope is that this will lead to better generalized performance on unseen instances.

4 Augmented Dependency Trees

Our task is to detect and classify relations between entities in text. We assume that entity tagging has been performed; so to generate potential relation instances, we iterate over all pairs of entities occurring in the same sentence. For each entity pair, we create a *dependency tree* (described be-

Feature	Example
word	*troops, Tikrit*
part-of-speech (*24 values*)	NN, NNP
general-pos (*5 values*)	noun, verb, adj
chunk-tag	NP, VP, ADJP
entity-type	person, geo-political-entity
entity-level	name, nominal, pronoun
Wordnet hypernyms	*social group, city*
relation-argument	ARG_A, ARG_B

Table 3: List of features assigned to each node in the dependency tree.

low) representing this instance. Given a labeled training set of potential relations, we define a *tree kernel* over dependency trees which we then use in an SVM to classify test instances.

A dependency tree is a representation that denotes grammatical relations between words in a sentence (Figure 1). A set of rules maps a parse tree to a dependency tree. For example, subjects are dependent on their verbs and adjectives are dependent on the nouns they modify. Note that for the purposes of this paper, we do not consider the link labels (e.g. "object", "subject"); instead we use only the dependency structure. To generate the parse tree of each sentence, we use MXPOST, a maximum entropy statistical parser[1]; we then convert this parse tree to a dependency tree. Note that the left-to-right ordering of the sentence is maintained in the dependency tree only among siblings (i.e. the dependency tree does not specify an order to traverse the tree to recover the original sentence).

For each pair of entities in a sentence, we find the smallest common subtree in the dependency tree that includes both entities. We choose to use this subtree instead of the entire tree to reduce noise and emphasize the local characteristics of relations. We then represent each node of the tree as a feature vector. For example, in addition to the word itself, we include the features in Table 3. The relation-argument feature specifies whether an entity is the first or second argument in a relation. This is required to learn asymmetric relations (e.g. X OWNS Y).

Formally, a relation instance is a dependency tree T with nodes $\{t_0 \dots t_n\}$. We use t_i to refer both to a tree node and to the feature vector that represents it, $t_i = \{v_1 \dots v_d\}$. We refer to the jth child of node t_i as $t_i[j]$, and we denote the set of all children of node t_i as $t_i[\mathbf{c}]$. We reference a subset $\mathbf{j}$ of children of t_i by $t_i[\mathbf{j}] \in t_i[\mathbf{c}]$. Finally, we refer to the parent of node t_i as $t_i.p$.

From the example in Figure 1, $t_0[1] = t_2$, $t_0[0,1] = \{t_1, t_2\}$, and $t_1.p = t_0$.

[1] http://www.cis.upenn.edu/~adwait/statnlp.html

5 Tree kernels for dependency trees

We now define a kernel function for dependency trees. The tree kernel is a function $K(T_1, T_2)$ that returns a normalized, symmetric similarity score in the range $(0, 1)$ for two trees T_1 and T_2. We define a slightly more general version of the kernel described by Zelenko et al. (2003).

We first define two functions over tree nodes: a matching function $m(t_i, t_j) \in \{0, 1\}$ and a similarity function $s(t_i, t_j) \in (0, \infty]$. Let the feature vector representing node $t_i = \{v_1 \dots v_d\}$ consist of two possibly overlapping subsets $t_i^m \subseteq t_i$ and $t_i^s \subseteq t_i$. We use t_i^m in the matching function and t_i^s in the similarity function. We define

$$m(t_i, t_j) = \begin{cases} 1 & \text{if } t_i^m = t_j^m \\ 0 & \text{otherwise} \end{cases}$$

and

$$s(t_i, t_j) = \sum_{v_q \in t_i^s} \sum_{v_r \in t_j^s} C(v_q, v_r)$$

where $C(v_q, v_r)$ is some compatibility function between two feature values. For example, in the simplest case where

$$C(v_q, v_r) = \begin{cases} 1 & \text{if } v_q = v_r \\ 0 & \text{otherwise} \end{cases}$$

$s(t_i, t_j)$ returns the number of feature values in common between nodes t_i^s, t_j^s.

We can think of the distinction between functions $m(t_i, t_j)$ and $s(t_i, t_j)$ as a way to discretize the similarity between two nodes. If none of the features in t_i^m match, then we declare the two nodes completely dissimilar. If t_i^m *does* match t_j^m, then we proceed to compute the similarity $s(t_i, t_j)$. Thus, restricting nodes by $m(t_i, t_j)$ is a way to prune the search space of matching subtrees, as shown below.

For two dependency trees T_1, T_2, with root nodes r_1 and r_2, we define the tree kernel $K(T_1, T_2)$ as follows:

$$K(T_1, T_2) = \begin{cases} 0 & \text{if } m(r_1, r_2) = 0 \\ s(r_1, r_2) + & \\ \qquad K_c(r_1[\mathbf{c}], r_2[\mathbf{c}]) & \text{otherwise} \end{cases}$$

where K_c is a kernel function over children. Let $\mathbf{a}$ and $\mathbf{b}$ be sequences of indices such that $\mathbf{a}$ is a sequence $a_1 \leq a_2 \leq \ldots \leq a_n$, and likewise for $\mathbf{b}$. Let $d(\mathbf{a}) = a_n - a_1 + 1$ and $l(\mathbf{a})$ be the length of $\mathbf{a}$. Then we have $K_c(t_i[\mathbf{c}], t_j[\mathbf{c}]) =$

$$\sum_{\mathbf{a}, \mathbf{b}, l(\mathbf{a}) = l(\mathbf{b})} \lambda^{d(\mathbf{a})} \lambda^{d(\mathbf{b})} K(t_i[\mathbf{a}], t_j[\mathbf{b}])$$

The constant $0 < \lambda < 1$ is a decay factor that penalizes matching subsequences that are spread out within the child sequences. See Zelenko et al. (2003) for a proof that K is kernel function.

Intuitively, whenever we find a pair of matching nodes, we search for all *matching subsequences* of the children of each node. A matching subsequence of children is a sequence of children $\mathbf{a}$ and $\mathbf{b}$ such that $m(a_i, b_i) = 1$ $(\forall i < n)$. For each matching pair of nodes (a_i, b_i) in a matching subsequence, we accumulate the result of the similarity function $s(a_i, b_j)$ and then recursively search for matching subsequences of *their* children $a_i[\mathbf{c}]$, $b_j[\mathbf{c}]$.

We implement two types of tree kernels. A *contiguous* kernel only matches children subsequences that are uninterrupted by non-matching nodes. Therefore, $d(\mathbf{a}) = l(\mathbf{a})$. A *sparse* tree kernel, by contrast, allows non-matching nodes within matching subsequences.

Figure 2 shows two relation instances, where each node contains the original text plus the features used for the matching function, $t_i^m = \{$general-pos, entity-type, relation-argument$\}$. ("NA" denotes the feature is not present for this node.) The contiguous kernel matches the following substructures: $\{t_0[0], u_0[0]\}$, $\{t_0[2], u_0[1]\}$, $\{t_3[0], u_2[0]\}$. Because the sparse kernel allows non-matching nodes, it matches an additional substructure $\{\{t_0[0, *, 2], u_0[0, *, 1]\}$, where $(*)$ indicates an arbitrary number of non-matching nodes.

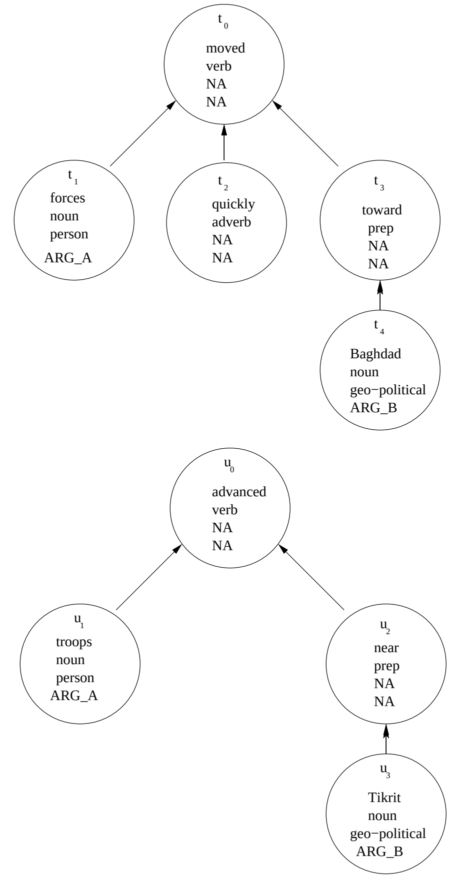

Figure 2: Two instances of the NEAR relation.

Zelenko has shown the contiguous kernel to be computable in $O(mn)$ and the sparse kernel in $O(mn^3)$, where m and n are the number of children in trees T_1 and T_2 respectively.

6 Experiments

We extract relations from the Automatic Content Extraction (ACE) corpus provided by the National Institute for Standards and Technology (NIST). The data consists of about 800 annotated text documents gathered from various newspapers and broadcasts. Five entities have been annotated (PERSON, ORGANIZATION, GEO-POLITICAL ENTITY, LOCATION, FACILITY), along with 24 types of relations (Table 2). As noted from the distribution of relationship types in the training data (Figure 3), data imbalance and sparsity are potential problems.

In addition to the contiguous and sparse tree kernels, we also implement a bag-of-words kernel, which treats the tree as a vector of features over nodes, disregarding any structural information. We also create *composite kernels* by combining the sparse and contiguous kernels with the bag-of-words kernel. Joachims et al. (2001) have shown that given two kernels K_1, K_2, the composite kernel $K_{12}(x_i, x_j) = K_1(x_i, x_j) + K_2(x_i, x_j)$ is also a kernel. We find that this composite kernel improves performance when the Gram matrix G is sparse (i.e. our instances are far apart in the kernel space).

The features used to represent each node are shown in Table 3. After initial experimentation, the set of features we use in the matching function is $t_i^m = \{$general-pos, entity-type, relation-argument$\}$, and the similarity function examines the remaining features.

In our experiments we tested the following five kernels:

$$\begin{aligned} K_0 &= \text{sparse kernel} \\ K_1 &= \text{contiguous kernel} \\ K_2 &= \text{bag-of-words kernel} \\ K_3 &= K_0 + K_2 \\ K_4 &= K_1 + K_2 \end{aligned}$$

We also experimented with the function $C(v_q, v_r)$, the compatibility function between two feature values. For example, we can increase the importance of two nodes having the same Wordnet hypernym[2]. If v_q, v_r are hypernym features, then we can define

$$C(v_q, v_r) = \begin{cases} \alpha & \text{if } v_q = v_r \\ 0 & \text{otherwise} \end{cases}$$

When $\alpha > 1$, we increase the similarity of nodes having the same hypernym. We tested a number of weighting schemes, but did not obtain a set of weights that produced consistent significant improvements. See Section 8 for alternate approaches to setting C.

Table 4 shows the results of each kernel within an SVM. (We augment the LibSVM[3] implementation to include our dependency tree kernel.) Note

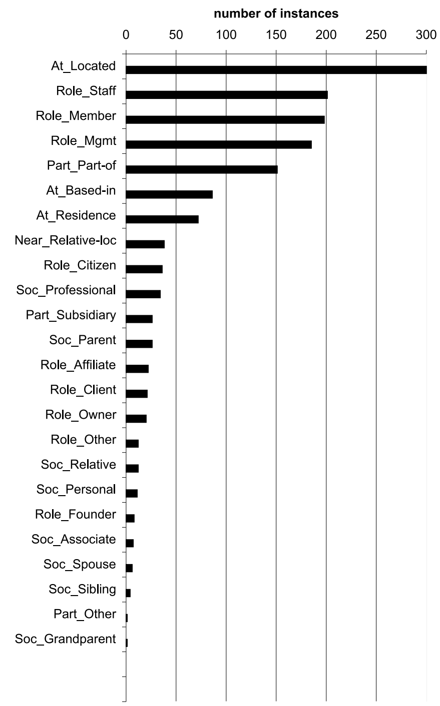

Figure 3: Distribution over relation types in training data.

that, although training was done over all 24 relation subtypes, we evaluate only over the 5 high-level relation types. Thus, classifying a RESIDENCE relation as a LOCATED relation is deemed correct[4]. Note also that K_0 is not included Table 4 because of burdensome computational time. While precision is adequate, recall is low. This is a result of the aforementioned class imbalance – very few of the training examples are relations, so the classifier is less likely to identify a testing instances as a relation. Because we treat every pair of mentions in a sentence as a possible relation, our training set contains fewer than 15% positive relation instances.

To remedy this, we retrain each SVMs for a binary classification task. Here, we detect, but do not classify, relations. This allows us to combine all positive relation instances into one class, which provides us more training samples to estimate the class boundary. We then threshold our output to

[2] http://www.cogsci.princeton.edu/˜wn/

[3] http://www.csie.ntu.edu.tw/˜cjlin/libsvm/

[4] This is to compensate for the small amount of training data for many classes.

	Avg. Prec.	Avg. Rec.	Avg. F1
K_1	69.6	25.3	36.8
K_2	47.0	10.0	14.2
K_3	68.9	24.3	35.5
K_4	**70.3**	**26.3**	**38.0**

Table 4: Kernel performance comparison.

	Prec.	Rec.	F1
K_0	–	–	–
K_0 (B)	83.4	45.5	58.8
K_1	91.4	37.1	52.8
K_1 (B)	84.7	49.3	62.3
K_2	92.7	10.6	19.0
K_2 (B)	72.5	40.2	51.7
K_3	91.3	35.1	50.8
K_3 (B)	80.1	49.9	61.5
K_4	91.8	37.5	53.3
K_4 (B)	81.2	**51.8**	**63.2**

Table 5: Relation *detection* performance. (B) denotes binary classification.

achieve an optimal operating point. As seen in Table 5, this method of relation detection outperforms that of the multi-class classifier.

We then use these binary classifiers in a cascading scheme as follows: First, we use a classifier to detect possible relations. Then, we use a classifier trained only on positive relation instances to classify each predicted relation. These results are shown in Table 6.

The first result of interest is that the sparse tree kernel, K_0, does not perform as well as the contiguous tree kernel, K_1. Suspecting that noise was introduced by the non-matching nodes allowed in the sparse tree kernel, we performed the experiment with different values for the decay factor $\lambda = \{.9, .5, .1\}$, but obtained no improvement.

The second result of interest is that all tree kernels outperform the bag-of-words kernel, K_2, most noticeably in recall performance, implying that the structural information the tree kernel provides is extremely useful for relation extraction.

D	**C**	Avg. Prec.	Avg. Rec.	Avg. F1
K_0	K_0	66.0	29.0	40.1
K_1	K_1	66.6	32.4	43.5
K_2	K_2	62.5	27.7	38.1
K_3	K_3	**67.5**	34.3	45.3
K_4	K_4	67.1	**35.0**	**45.8**
K_1	K_4	67.4	33.9	45.0
K_4	K_1	65.3	32.5	43.3

Table 6: Results on the cascading classification. **D** and **C** denote the kernel used for relation detection and classification, respectively.

7 Conclusions

We have shown that using a dependency tree kernel for relation extraction provides a vast improvement over a bag-of-words kernel. While the dependency tree kernel appears to perform well at the task of classifying relations, recall is still relatively low. *Detecting* relations is a difficult task for a kernel method because the set of all *non-relation* instances is extremely heterogeneous, and is therefore difficult to characterize with a similarity metric. An improved system might use a different method to detect candidate relations and then use this kernel method to classify the relations.

8 Future Work

The most immediate extension is to automatically learn the feature compatibility function $C(v_q, v_r)$. A first approach might use tf-idf to weight each feature. Another approach might be to calculate the information gain for each feature and use that as its weight. A more complex system might learn a weight for each pair of features; however this seems computationally infeasible for large numbers of features.

One could also perform latent semantic indexing to collapse feature values into similar "categories" — for example, the words "football" and "baseball" might fall into the same category. Here, $C(v_q, v_r)$ might return α_1 if $v_q = v_r$, and α_2 if v_q and v_r are in the same category, where $\alpha_1 > \alpha_2 > 0$. Any method which provides a "soft" match between feature values will sharpen the granularity of the kernel and enhance its modeling power.

References

Eugene Agichtein and Luis Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*.

Sergey Brin. 1998. Extracting patterns and relations from the world wide web. In *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT'98*.

M. Collins and N. Duffy. 2002. Convolution kernels for natural language. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

N. Cristianini and J. Shawe-Taylor. 2000. *An introduction to support vector machines*. Cambridge University Press.

Chad M. Cumby and Dan Roth. 2003. On kernel methods for relational learning. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*. AAAI Press.

K. Fukunaga. 1990. *Introduction to Statistical Pattern Recognition*. Academic Press, second edition.

D. Haussler. 1999. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, University of California, Santa Cruz.

Thorsten Joachims, Nello Cristianini, and John Shawe-Taylor. 2001. Composite kernels for hypertext categorisation. In Carla Brodley and Andrea Danyluk, editors, *Proceedings of ICML-01, 18th International Conference on Machine Learning*, pages 250–257, Williams College, US. Morgan Kaufmann Publishers, San Francisco, US.

Huma Lodhi, John Shawe-Taylor, Nello Cristianini, and Christopher J. C. H. Watkins. 2000. Text classification using string kernels. In *NIPS*, pages 563–569.

A. McCallum and B. Wellner. 2003. Toward conditional models of identity uncertainty with application to proper noun coreference. In *IJCAI Workshop on Information Integration on the Web*.

S. Miller, H. Fox, L. Ramshaw, and R. Weischedel. 2000. A novel use of statistical parsing to extract information from text. In *6th Applied Natural Language Processing Conference*.

H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. 2002. Identity uncertainty and citation matching.

Dan Roth and Wen-tau Yih. 2002. Probabilistic reasoning for entity and relation recognition. In *19th International Conference on Computational Linguistics*.

Sam Scott and Stan Matwin. 1999. Feature engineering for text classification. In *Proceedings of ICML-99, 16th International Conference on Machine Learning*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada.

Vladimir Vapnik. 1998. *Statistical Learning Theory*. Whiley, Chichester, GB.

D. Zelenko, C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *Journal of Machine Learning Research*, pages 1083–1106.