On Kernel Methods for Relational Learning

Chad Cumby CUMBY@UIUC.EDU
Dan Roth DANR@UIUC.EDU
Department of Computer Science University of Illinois, Urbana, IL 61801 USA

Abstract

Kernel methods have gained a great deal of popularity in the machine learning community as a method to learn indirectly in high-dimensional feature spaces. Those interested in relational learning have recently begun to cast learning from structured and relational data in terms of kernel operations.

We describe a general family of kernel functions built up from a description language of limited expressivity and use it to study the benefits and drawbacks of kernel learning in relational domains. Learning with kernels in this family directly models learning over an expanded feature space constructed using the same description language. This allows us to examine issues of time complexity in terms of learning with these and other relational kernels, and how these relate to generalization ability. The tradeoffs between using kernels in a very high dimensional implicit space versus a restricted feature space, is highlighted through two experiments, in bioinformatics and in natural language processing.

1. Introduction

Recently, much interest has been generated in the machine learning community on the subject of learning from relational and structured data via propositional learners (Kramer et al., 2001; Cumby & Roth, 2003a). Examples of relational learning problems include learning to identify functional phrases and named entities from structured parse trees in natural language processing (NLP), learning to classify molecules for mutagenicity from atom-bond data in drug design, and learning a policy to map goals to actions in planning domains. At the same time, work on SVMs and Perceptron type algorithms has generated interest in kernel methods to simulate learning in high-dimensional feature spaces while working with the original low-dimensional input data.

Haussler's work on convolution kernels (Haussler, 1999) introduced the idea that kernels could be built to work with discrete data structures iteratively from kernels for smaller composite parts. These kernels followed the form of a generalized sum over products – a generalized convolution. Kernels were shown for several discrete datatypes including strings and rooted trees, and more recently (Collins & Duffy, 2002) developed kernels for datatypes useful in many NLP tasks, demonstrating their usefulness with the Voted Perceptron algorithm (Freund & Schapire, 1998).

While these past examples of relational kernels are formulated separately to meet each problem at hand, we seek to develop a flexible mechanism for building kernel functions for many structured learning problems - based on a unified knowledge representation. At the heart of our approach is a definition of a relational kernel that is specified in a "syntax-driven" manner through the use of a description language. (Cumby & Roth, 2002) introduced a feature description language and have shown how to use propositional classifiers to successfully learn over structured data, and produce relational representation, in the sense that different data instantiations yield the same features and have the same weights in the linear classifier learned. There, as in (Roth & Yih, 2001), this was done by significantly blowing up the relational feature-space.

Building on the abovementioned description language based approach, this paper develops a corresponding family of parameterized kernel functions for structured data. In conjunction with an SVM or a Perceptron-like learning algorithm, our parameterized kernels can simulate the exact features generated in the blown up space to learn a classifier, directly from the original structured data. From among several ways to define the distance between structured domain elements we follow (Khardon et al., 2001) in choosing a definition that provides exactly the same classifiers produced by Perceptron, if we had run it on the blown up discrete feature space, rather than directly on the structured data. The parameterized kernel allows us to flexibly

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

define features over structures (or, equivalently, a metric over structures). At the same time, it allows us to choose the degree to which we want to restrict the size of the expanded space, which affects the degree of efficiency gained or lost - as well as the generalization performance of the classifier - as a result of using it.

Along these lines, we then study time complexity and generalization tradeoffs between working in the expanded feature space and using the corresponding kernels, and between different kernels, in terms of the expansions they correspond to. We show that, while kernel methods provide an interesting and often more comprehensible way to view the feature space, computationally, it is often not recommended to use kernels over structured data. This is especially clear when the number of examples is fairly large relative to the data dimensionality.

The remainder of the paper is organized as follows: We first introduce the use of kernel functions in linear classification and the *kernel Perceptron* algorithm (Khardon et al., 2001). Sec. 3 presents our approach to relational features, which form the higher-dimensional space we seek to simulate. Sec. 4 introduces our kernel function for relational data. Sec. 5 discusses complexity tradeoffs surrounding the kernel Perceptron and standard feature-based Perceptron and the issue of generalization. In Sec. 6 we validate our claims by applying the kernel Perceptron algorithm with our enhanced kernel to two problems where a structured feature space is essential.

2. Kernels & Kernel Perceptron

Most machine learning algorithms make use of feature based representations. A domain element is transformed into a collection of Boolean features, and a labeled example is given by $< x, l > \in \{0,1\}^n \times \{-1,1\}$.

In recent years, it has become very common, especially in large scale applications in NLP and computer vision, to use learning algorithms such as variations of Perceptron and Winnow (Novikoff, 1963; Littlestone, 1988) that use representations that are linear over their feature space (Roth, 1998). In these cases, working with features that directly represent domain elements may not be expressive enough and there are standard ways of enhancing the capabilities of such algorithms. A typical way which has been used in the NLP domain (Roth, 1998; Roth, 1999) is to expand the set of basic features $x_1, \ldots, x_n$ using *feature functions* $\chi_i : \{x_1, \ldots x_n\} \rightarrow \{0,1\}$, $i \in I$. These features functions could be, for example, conjunctions such as $x_1\overline{x_3}x_4$ (that is, the feature is evaluated to 1 when the conjunction is satisfied on the example, and to 0 otherwise). Once this expansion is done, one can use these expanded higher-dimensional examples, in which each feature function plays the role of a basic feature, for learning. This approach clearly leads to an increase in expressiveness and thus may improve performance. However, it also dramatically increases the number of features (from n to $|I|$; e.g., to 3^n if all conjunctions are used, $O(n^k)$ if conjunctions of size k are used) and thus may adversely affect both the computation time and convergence rate of learning.

Perceptron is a well known on-line learning algorithm that makes use of the aforementioned feature based representation of examples. Throughout its execution Perceptron maintains a weight vector $w \in \Re^n$ which is initially $(0, \ldots, 0)$. Upon receiving an example $x \in \{0,1\}^n$, the algorithm predicts according to the linear threshold function $w \cdot x \geq 0$. If the prediction is 1 and the label is -1 then the vector w is set to $w - x$, while if the prediction is -1 and the label is 1 then w is set to $w + x$. No change is made if the prediction is correct.

It is easily observed, and well known, that the hypothesis w of the Perceptron is a $\pm$ sum of the previous examples on which prediction mistakes were made. Let $L(x) \in \{-1,1\}$ denote the label of example x; then $w = \sum_{v \in M} L(v)v$ where M is the set of examples on which the algorithm made a mistake. Thus the prediction of Perceptron on x is 1 iff $w \cdot x = (\sum_{v \in M} L(v)v) \cdot x = \sum_{v \in M} L(v)(v \cdot x) \geq 0$.

For an example $x \in \{0,1\}^n$ let $\phi(x)$ denote its transformation into an enhanced feature space, $\phi(x) = \{\chi_i(x)\}_{i \in I}$.

To run the Perceptron algorithm over the enhanced space we must predict 1 iff $w^\phi \cdot \phi(x) \geq 0$ where w^ϕ is the weight vector in the enhanced space; from the above discussion this holds iff $\sum_{v \in M} L(v)(\phi(v) \cdot \phi(x)) \geq 0$.

Denoting

$$K(v, x) = \phi(v) \cdot \phi(x) \tag{1}$$

this holds iff $\sum_{v \in M} L(v)K(v, x) \geq 0$. We call this version of Perceptron, which uses a kernel function to expand the feature space and thus enhances the learning abilities of Perceptron, a *kernel Perceptron* (Khardon et al., 2001).

We have shown a bottom up construction of the "kernel trick". This way, we never need to construct the enhanced feature space explicitly; we need only be able to compute the kernel function $K(v, x)$ efficiently. This trick can be applied to any algorithm whose prediction is a function of inner products of examples; see e.g. (Cristianini & Shawe-Taylor, 2000) for a discussion. In principle, one can define the kernel $K(x, v)$ in different ways, as long as it satisfies some conditions. In this paper we will concentrate on a definition that follows

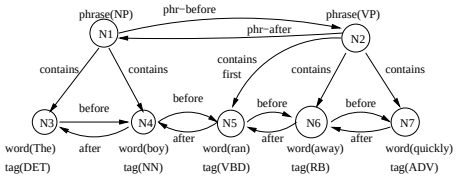

Figure 1. A concept graph for a partial parse of a sentence.

the discussion above. We will first define a feature space. Then define a collection of features functions and a mapping ϕ to an enhanced feature space. We will then show that our kernel definition satisfies Eq.1. The discussion above implies that running kernel Perceptron yields identical results to running Perceptron on the enhanced features space.

3. Relational Features

In order to learn in structured domains such as natural language, object recognition, and computational biology, one must decide what representation is to be used for both the concept to be learned, and for the instances that will be learned from. Given the intractability of traditional Inductive Logic Programming (ILP) approaches under many conditions, recent approaches attempt to develop ways that use efficient propositional algorithms over relational data. One tactic that has shown itself to be particularly useful is the method of "propositionalization" (Kramer et al., 2001; Cumby & Roth, 2003a). This method suggests to represent relational data in the form of quantified propositions (Khardon et al., 1999), which can be used with standard propositional and probabilistic learners. This technique allows the expansion of the space of possible propositional features to include many structured features defined over basic features abstracted from the input data instances. One method of performing this expansion of relational features is through the notion of a Feature Description Logic.

By the method detailed in (Cumby & Roth, 2002), this Description Logic allows one to define "types" of features, which through an efficient generation function, are instantiated by comparing the feature definition against data instances. These data instances are in general any type of structured data, represented by a graph-based data structure known as a *concept graph*.

For an example of a concept graph, consider the sentence represented by the dependency graph in Fig. 1. This digraph contains nodes to represent each word, along with two nodes to represent the noun and verb phrase present in the sentence. In general, each node is labeled with attribute information about some individual, and each edge is labeled with relational information about two individuals. The description logic, as defined below, provides a framework for representing the same semantic individuals as are represented by nodes in a concept graph.

Definition 1 *An FDL* description *over the* attribute *alphabet* $Attr = \{a_1, ..., a_n\}$*, the* value *alphabet* $Val = v_1, ..., v_n$*, and the* role *alphabet* $Role = \{r_1, ..., r_n\}$ *is defined inductively as follows:*

1. *For an attribute symbol* a_i *and a value symbol* v_j*,* $a_i(v_j)$ *is a description called a* sensor*.* a_i *is also a description, an* existential sensor*. (A sensor represents the set* $x \in X$ *for which* $a_i(x, v_j)$ *is true. An existential sensor represents the set* $x \in X$ *s.t.* $\exists v_j$*, s.t.* $a_i(x, v_j)$ *is true.)*
2. *If* D *is a description and* r_i *is a role symbol, then* **(**r_i **D)** *is a* role *description. (Represents the set* $x \in X$ *such that* $r_i(x, y)$ *is true iff for* $y \in X$*,* y *is described by* D*.)*
3. *If* $D_1, ..., D_n$ *are descriptions, then (***AND** $D_1, ..., D_n$*) is a description. (The conjunction of several descriptions.)*

In order to generate useful features from the structured instance above we could, for example, define a description such as: (**AND** phrase (contains word)). Each feature to be generated from it is a Boolean valued function indexed by a syntactic description. Below are features generated through the Feature Generating Function (FGF) mechanism described in (Cumby & Roth, 2002). To do so, the generating description is matched against each node of the given instance graph. Feat. (1) below means essentially, "In the given data instance, output 1 if $\exists x, y \in X$ such that $phrase(x, NP) \wedge contains(x, y) \wedge word(y, The)$".

1. (**AND** phrase(NP) (contains word(The)))
2. (**AND** phrase(NP) (contains word(boy)))
3. (**AND** phrase(VP) (contains word(ran)))
4. (**AND** phrase(VP) (contains word(away)))
5. (**AND** phrase(VP) (contains word(quickly)))

We note that it is an easy matter to move from a pure Boolean representation of features to a positive integer valued representation, e.g., by associating the number of times each feature substructure occurs in a given instance. This change of feature representation does not change the operation of the Perceptron algorithm and is related to our kernel construction in Sec. (4).

The generation of features in such a manner produces a feature space polynomial in the size of the original input space with the degree being the generating description size. This is attractive in terms of complexity of generation and also potentially in terms of the ability to learn efficiently in such spaces. We return to this topic in Sec. (5).

4. Relational Kernels

We now define a family of kernel functions based on the Feature Description Language introduced. We utilize the description language introduced in Sec. (3) with a different purpose - to define the operation of a kernel function - but still with the aim of exploiting the syntax of the language to determine the shape of the feature-space. The definition of our kernel functions uses the formal notion of the FDL concept graphs introduced earlier, whose definition we summarize here:

An FDL concept graph is a labeled directed acyclic graph $G = G(N, E, l_N(*))$, where N is a set of nodes, $E \subseteq (N \times N \times Role)$ a set of labeled edges (with role symbols as labels) and $l_N(*)$ a function that maps each node in N to a set of sensor descriptions associated with it. A rooted concept graph is specified by designating a node $n_0 \in N$ as the root of the graph.

With these two definitions we define the operation of a family of kernels parameterized by an FDL description. We call this parameter the *generating* description. Each kernel takes as input two directed acyclic concept graphs, which may represent many types of structured data. Each outputs a real valued number representing the similarity of the two concept graphs with respect to the generating description. The family of kernels is "syntax-driven" in the sense that the input description specifies the kernel's operation.

Definition 2 *Let $\mathcal{D}$ be the space of FDL descriptions, and $\mathcal{G}$ the space of all concept graphs. The family of functions $\mathcal{K} = \{k_D | D \in \mathcal{D}\}$ is defined inductively as:*

$$k_D(G_1, G_2) = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} k_D(n_1, n_2)$$

1. *If D is a sensor description of the form $s(v)$ and $s(v) \in l_{N_1}(n_1) \cap l_{N_2}(n_2)$, then $k_D(n_1, n_2) = 1$.*
2. *If D is an existential sensor description of the form s, and $\exists V \subseteq Val$ s.t. $\forall v \in V$ $s(v) \in l_{N_1}(n_1) \cap l_{N_2}(n_2)$, then $k_D(n_1, n_2) = |V|$.*
3. *If D is a role description of the form $(r\ D')$: Let N_1 be the set of all nodes $\{n'_1 \in N | (n_1, n'_1, r) \in E\}$, and let N_2 be the set of all nodes $\{n'_2 \in N | (n_2, n'_2, r) \in E\}$. Then $k_D(n_1, n_2) = \sum_{n'_1} \sum_{n'_2} k_{D'}(n'_1, n'_2)$.*
4. *If D is a description of the form (**AND** D_1 D_2 $\ldots D_n$), with l_i repetitions of any D_i, then $k_D(n_1, n_2) = \prod_{i=1}^{n} \binom{k_{D_i}(n_1, n_2)}{l_i}$.*

Theorem 3 *For any FDL description D and for any two concept graphs G_1, G_2, the quantity output by $k_D(G_1, G_2)$ is equivalent to the dot product of the two feature vectors $\phi_D(G_2)$, $\phi_D(G_2)$ output by the feature generating function described in (Cumby & Roth, 2002). Thus for all D k_D necessarily defines a kernel function over the space of concept graphs $\mathcal{G}$.*

For a proof of Theorem 3 see (Cumby & Roth, 2003b).

We examplify the claim that k defines a kernel function by demonstrating that we can simulate two expanded feature spaces directly with our kernel construction. First of all, the technique can generalize the simple propositional case as considered in (Khardon et al., 2001) and also generalize learning with word or POS-tag n-grams of the type often considered in NLP applications, given that the words and tags are encoded as sensor descriptions appropriately.

In order to mimic the case of learning from simple Boolean bit vectors such as $[x_1 x_2 \ldots x_n] \in \{0,1\}^n$, we perform the following translation. For each such vector, define a mapping from each component x_i in $x \in \{0,1\}^n$ to a relation $s(x, v_i)$ where v_i corresponds to the truth value of x_i on x. We can then use an FDL concept graph consisting of a single node labeled with sensor descriptions corresponding to each $s(x, v_i)$ to represent x. At this point it becomes a simple matter to mimic the parameterized kernel of (Khardon et al., 2001) using our feature description kernel. Define the generating descriptions:

$$D_1 = s; D_2 = (\mathbf{AND} ss); D_3 = (\mathbf{AND} sss), \ldots$$

If we evaluate each $k_{D_i}(n_1, n_2)$ for two single-node instances n_1, n_2 as described above, we expect to capture each conjunction of literals up to size three. Let $same(n_1, n_2)$ denote the number of sensor descriptions present as labels of both n_1 and n_2. Then we have $\sum_{i=1}^{3} k_{D_i}(n_1, n_2) = same(n_1, n_2) + \binom{k_s(n_1,n_2)}{2} + \binom{k_s(n_1,n_2)}{3} = \sum_{i=1}^{3} \binom{same(n_1,n_2)}{i}$. This directly mirrors the parameterized kernel detailed in (Khardon et al., 2001). The expanded space feature vectors $\phi(n_1)$ $\phi(n_2)$ for nodes n_1 n_2 consist of all conjunctions of sensors up to size 3, and the dot product $\phi(n_1) \cdot \phi(n_2)$ is equal to the number of conjunctions active in both vectors. This quantity is equal to $\sum_i k_{D_i}(n_1, n_2)$.

The example of extracting k-conjunctions from bit vectors is markedly different from the example of generating n-gram type features as seen in the earlier example. In the case of n-grams, relational information is implicit in the manner in which combinations of words or other objects are chosen to be output. For example in the sentence: *The boy ran quickly*, the combination *The-boy* is a valid bigram whereas *boy-The* is not. Via our kernel definition, we can simulate the operation of a feature-space based algorithm on bigrams as follows.

Given the generating description

D = (**AND** word (before word))

along with two data instances represented as concept graphs G_1, G_2, where G_1 corresponds to the instance in Fig. 1 and G_2 to a similar instance for the sentence *The boy ran quickly*, we can observe the output of the function $k_D(G_1, G_2) = \sum_{n_1 \in G_1} \sum_{n_2 \in G_2} k_D(n_1, n_2)$. For four pairings of n_1 and n_2, $k_{word}(n_1, n_2)$ will be non-zero, corresponding to the pair of the nodes labeled with *word(The)*, *word(boy)*, *word(ran)*, and with *word(quickly)*. The output of the first pairing is $k_D(N^1_{The}, N^2_{The}) = k_{word}(N^1_{The}, N^2_{The}) \cdot k_{(before\ word)}(N^1_{The}, N^2_{The}) = 1 \cdot k_{word}(N^1_{boy}, N^2_{boy}) = 1 \cdot 1 = 1$. The output for the second pairing of $k_D(N^1_{boy}, N^2_{boy})$ is also 1 by a similar evaluation. The output of the third pairing is $k_D(N^1_{ran}, N^2_{ran}) = k_{word}(N^1_{ran}, N^2_{ran}) \cdot k_{(before\ word)}(N^1_{ran}, N^2_{ran}) = 1 \cdot k_{word}(N_{away}, N_{quickly}) = 1 * 0 = 0$. And the output for the fourth pairing is $k_D(N^1_{quickly}, N^2_{quickly}) = k_{word}(N^1_{quickly}, N^2_{quickly}) \cdot 0 = 1 \cdot 0 = 0$. Thus $k_D(G_1, G_2) = 1 + 1 + 0 = 2$.

If we were to expand the feature-space for G_1 using the description D and the method referred to in Section (3), we would have a feature vector $\phi(G_1)$ with the following features outputting 1 and all else 0:

- (**AND** word(The) (before word(boy)))
- (**AND** word(boy) (before word(ran)))
- (**AND** word(ran) (before word(away)))
- (**AND** word(away) (before word(quickly)))

Computing $\phi(G_2)$ in a similar manner produces a vector with these active features, and all else 0:

- (**AND** word(The) (before word(boy)))
- (**AND** word(boy) (before word(ran)))
- (**AND** word(ran) (before word(quickly)))

Computing the dot-product: $\phi(G_1) \cdot \phi(G_2) = 2$. Thus $k_D(G_1, G_2) = \phi(G_1) \cdot \phi(G_2)$, exemplifying Thm 3.

Before continuing, we present an extension to Def. 2 with its corresponding effects on the types of kernels included in the family defined in Def. 2, intended to incorporate the notion of *locality* along with *focus of interest*. The focus of interest for a given input concept graph is defined as a single distinguished node f.

Definition 4 *We extend the language of Def. 2 with the following rule and define it as FDL_{loc}: If D is a description, then (***IN***[n,r] D) is also a description, where $n \in \mathcal{Z}^+$ and $R \in Role$. This description denotes the set of individuals $x \in X$ described by D and represented by a node in a given input concept graph within n R-labeled edges from the given focus f. When $R = Role$, we write* **IN***[n] D).*

The family of kernels $\mathcal{K}$ is extended accordingly to include kernels $k_{D_{loc}}(G_1, G_2)$, with D_{loc} of the form (**IN**[n,r] D), and $D \in FDL$. In this case G_1, G_2 are concept graphs each with a given focus of interest f_1, f_2. As before $k_{K_{loc}}(G_1, G_2) = \sum_{n_1 \in N_1 \in G_1} \sum_{n_2 \in N_2 \in G_2} k_{D_{loc}}(n_1, n_2)$. However here $k_{D_{loc}}(n_1, n_2) = k_D(n_1, n_2)$ if both n_1, n_2 are within n r-labeled edges from f_1 and f_2 respectively, or else $k_{D_{loc}}(n_1, n_2) = 0$.

5. Complexity & Generalization

Our treatment of kernels for relational data aims at developing a clear understanding of when it becomes advantageous to use kernel methods over standard learning algorithms operating over some feature space. The common wisdom in propositional learning is that kernel methods will always yield better performance because they cast the data in an implicit high-dimensional space (but also see (Khardon et al., 2001; Ben-David & Simon, 2002)).

Propositionalization methods in relational learning have demonstrated separately that it is possible to learn relational concepts using a transformation of the input data. There, however, the transformation into a high-dimensional feature space is done explicitly through the construction of propositional features, as done in Sec. (3) using the Feature Generating Function. The discussion of the relational kernel showed that we can formulate a kernel function that simulates the updates performed by running Perceptron over the features generated this way. Conceivably then, we should be able to perform a learning experiment comparing the kernel method and the explicitly generated feature space, and expect to achieve the same results in testing. Given such a situation, the main difference between the two methods then becomes the expected running time involved. Thus a general analysis of the complexity of running the kernel method and of generating features and running over them seems warranted.

Given some description in our FDL D, and two concept graphs G_1, G_2, let t_1 be the time to evaluate $k_D(n_1, n_2)$ if n_1, n_2 are two nodes of G_1, G_2 respectively. Assuming all input concept graphs are equally likely, let g be the average number of nodes in a given graph. Since $k_D(G_1, G_2) = \sum_{n_1 \in G_1} \sum_{n_2 \in G_2} k_D(n_1, n_2)$, the complexity of evaluating $k_D(G_1, G_2)$ is proportional to $g^2 t_1$. Since for an arbitrary sequence $x_1 \ldots x_m$ of input graphs the kernel Perceptron algorithm could, in the worst case, make

$i-1$ mistakes on x_i, the overall running time for the algorithm given this kernel is $O(m^2g^2t_1)$.

To analyze the normal version of Perceptron, run over an equivalent feature-space generated by the FGF algorithm given in (Cumby & Roth, 2002) with the same description D, we first assume that the time to evaluate χ_D for a node of a given concept graph is proportional to t_2. The total time to generate features from m arbitrary concept graphs is then $O(mgt_2)$ with g as defined above. We assume that the feature vectors we work with are variable length vectors of only positive features (this is equivalent learning in the infinite attribute space (Blum, 1992) and is common in applications (Roth, 1998)). To run Perceptron over the resulting feature vectors, each time a mistake is made, we update each weight corresponding to an active feature in the current example. We could abstract the number of active features per example with an average, but in reality these updates take time proportional to the time spent in generating the features for each input graph. Thus the total running time is $O(mgt_2)$.

It is interesting to note that this result also applies to the specific situation of kernel learning from Boolean bit vectors. Consider the example given earlier of extracting conjunctions up to size k from bit vectors of the form $(x_1x_2\dots x_n), (y_1y_2\dots y_n) \in \{0,1\}^n$. In this case, as the representation of each vector is mapped to a single node concept graph, $g = 1$. t_1 is proportional to computing $\sum_{i=1}^{k} \binom{same(x,y)}{i}$, which is $O(n)$. For m examples the total time of learning with kernel Perceptron is then $O(m^2n)$. To compute the feature space of conjunctions up to size k explicitly takes $O(n^k)$, thus the total time for running standard Perceptron over the blown up feature space is $O(mn^k)$. If $m > n^{k-1}$, the complexity disadvantage of the kernel approach becomes apparent.

A similar tradeoff can be seen in terms of generalization ability. Our discussion shows that learning with kernel methods simulates learning with a highly expanded feature space. It is not surprising then, that recent work (Weston et al., 2000; Ben-David & Simon, 2002; Khardon et al., 2001), has shown that even margin-maximizing kernel learners may suffer from the curse of dimensionality; this happens in cases where the use of kernels is equivalent to introducing a large number of irrelevant features. Assuming that a subset of the features generated is expressive enough for a given problem, it is not hard to show that embedding the problem in an even higher dimensional space can only be harmful to generalization.

Previous methods that use structured kernels (e.g. (Collins & Duffy, 2002)) make use of an implicit feature space that is exponential in the size of the input representation, by defining a metric that depends on all "substructures" (as do standard polynomial kernels). Our kernel approach allows a transformations of the input representation to an implicitly larger propositional space; it does so, however, using a parameterized kernel, thus allowing control of this transformation so that it can be equivalent to a smaller propositional feature space. By specifying a kernel through a syntax-driven mechanism based on the relational structure of the input, we can actively attempt to decrease the number of irrelevant features introduced. The benefit to generalization will be shown experimentally in the next section.

Note that the focus of this analysis is on kernel Perceptron; the exact complexity implications for other kernel learners such as SVM, beyond the quadratic dependence on the number of examples, is not so clear. The implications for generalization, however, apply across all kernel learners.

6. Experimental Validation

We present two experiments in the domains of bioinformatics and NLP. Our goal is to demonstrate that by restricting the expanded feature space that learning takes place in - using the standard feature based learner, or kernel Perceptron - we benefit in terms of generalization, and thus accuracy. Our main comparison is performed against a kernel learner that implicitly constructs an exponentially larger feature space using a kernel based on (Collins & Duffy, 2002). As we can explicitly generate the smaller feature space used in the FDL approach, it is possible to verify our equivalence claim and compare with the performance of other propositional learners that do not admit kernels. The latter comparison is done using a variation of Winnow (Littlestone, 1988) implemented in the SNoW system (Carlson et al., 1999).

The first experiment addresses the problem of predicting mutagenicity in organic compounds, which has become a standard benchmark in ILP for 'propositionalization' methods (Srinivasan et al., 1996). In this problem, a set of 188 compounds is given in the form of atom and bond tuples corresponding to the atoms in each molecule with their properties and links to other atoms. Additionally, a set of label tuples is given to indicate whether each molecule is mutagenic.

We map each compound to a concept graph, constructing nodes for each atom tuple labeled with sensor descriptions of the form *atom-elt*(v) for the element type, *atom-type*(v) for the atom type and *atom-chrg*(v) for the partial charge. We construct edges for each bond tuple and construct a single node for the compound

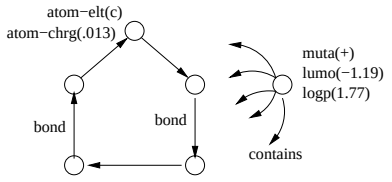

Figure 2. Concept graph for fragment of mutagenesis domain element

overall - connected to each atom and labeled with the compound's mutagenicity, *lumo*, and *logP* values.

We first perform classification using an expanded feature space generated by taking the sum of the kernels $k_{D_1}, k_{D_2}, k_{D_3}$, with the generating descriptions D_1, D_2, D_3 as described below:

- D_1 = lumo
- D_2 = logP
- D_3 = (**AND** atom-elt atom-chrg (bond (AND atom-elt atom-chrg (bond ...)))) *up to nine conjuncts*

Since kernels are composable under addition, the overall kernel is valid. We introduce the second experimental setup before continuing with results.

The next experiment deals with the natural language processing task of tagging Named Entities. Given a natural language sentence, we wish to determine which phrases in it correspond entities such as *locations, organizations* or *persons*. In this experiment, the MUC-7 dataset of sentences with labeled entities was used. Rather than focusing on the tagging problem of determining whether a word is in some entity or not, we instead attempt to classify what entity a given phrase corresponds to out of these three. we first convert the raw MUC data into a chunked form (with subsets of words marked as noun phrases, verb phrases, *etc*) and map this form to concept graphs as in Fig. 1.

We extracted a set of of 4715 training phrases and 1517 test phrases, and once again trained a classifier based on a restrictively expanded feature space as detailed in Sec. (3) and Sec. (4). The generating descriptions used to determine the types of features include:

- (IN[0] (first (AND word tag (after word))))
- (IN[0] (phr-before (first word)))
- (IN[0] (phr-before (first tag)))
- (IN[0] (phr-before phrase))
- (IN[0] (phr-after phrase))
- (IN[0] (contains (AND word tag)))

For each input graph, several noun phrase nodes may be present. Thus each instance is presented to the feature generating process/kernel learner several times, each time with a different phrase node designated as the *focus of interest*. The above descriptions reference this focus as mentioned in Sec. (4).

As we are trying to predict from a set of k category labels for each phrase, we are faced with a multi-class classification problem. We used a standard 1-vs-all method, with a winner take all gate for testing.

In each experiment we also trained a classifier using kernel Perceptron with a modified kernel based on the parse tree kernel given in (Collins & Duffy, 2002). This kernel tracks the number of common subtrees descending from the root node, which is designated to be the 'molecule' node in each example compound in the mutagenesis experiment, and the current phrase focus of interest node in the named entity experiment. Down-weighting of larger subtrees is performed also as in (Collins & Duffy, 2002). As our concept graphs may be cyclic, a termination rule is added to stop traversal of each graph if a node previously encountered on a traversal path is touched again. We show below the results of training and evaluating each type of classifier. For mutagenesis we train with 10-fold cross validation on the set of 188 compounds with 12 rounds of training for each fold. For the Named Entity task we train using 5 rounds of training over the entire training set. Results in terms of accuracy[1] are shown.

	SNoW (Winnow)	FDL kernel	All subtrees
mutagenesis	88.6%	85.4%	71.3%
NE class	75.41%	76.6%	52.9%

The main comparison in the above table is between the feature space represented in both col. 1 and 2, and the one represented in col. 3. In both experiments the classifiers learned over the FDL induced feature space perform markedly better than the the one learned over the 'all-subtrees' feature space. Col. 1 is the outcome of running a different classifier on an explicit feature space equivalent to the FDL of col. 2. It shows that using the explicitly propositionalized feature space, we can use other learning algorithms, not amenable to kernels, which might perform slightly better.

For the mutagenesis task the FDL approach encodes the typical features used in other ILP evaluations, indicating that many ILP tasks, where the relational data can be translated into our graph-based struc-

[1] In the mutagenesis case the data set is fairly balanced and results are usually reported in terms of accuracy (Srinivasan et al., 1996). For the NE case, each example corresponds to a valid NE, and the set is balanced (3:5:7) so micro and macro averaging are fairly close. The difference between Col.3 and Col. 1 and 2 is statistically significant.

ture (Cumby & Roth, 2002), can be addressed this way. The named entity example abstracts away the important task of determining which phrases correspond to valid entities, and performs sub-optimally in terms of classifying the entities correctly due to not exploiting richer features that can be used in this task. However, our goal here in the classification tasks is to exhibit the benefit of the FDL approach over the "all-subtrees" kernel. Namely, that by developing a parameterized kernel which restricts the range of structural features produced, we avoid over-fitting due to a large number of irrelevant features.

7. Conclusion

We have presented a new approach to constructing kernels for learning from structured data, through a description language of limited expressivity. This family of kernels is generated in a syntax-driven way parameterized by descriptions in the language. Thus, it highlights the relationship between an explicitly expanded feature space constructed using this language, and the implicitly simulated feature space that learning takes place in when using kernel based algorithms such as kernel Perceptron or SVM. In some cases, it is more efficient to learn in the implicitly expanded space, when this space may be exponential in the size of the input example, or infinite. However, we have shown that an expanded space of much higher dimensionality can degrade generalization relative to one restricted by the use of our syntactically determined kernel.

By studying the relationship between these explicit and implicitly simulated feature spaces, and the computational demands of kernel based algorithms such as kernel Perceptron, we have highlighted the cases in which, contrary to popular belief, working in the explicit feature space with the standard learning algorithms may be beneficial over kernels.

Acknowledgments: This research is supported by NSF grants ITR-IIS-0085836, ITR-IIS-0085980 and IIS-9984168 and an ONR MURI Award.

References

Ben-David, S., & Simon, N. E. U. H. (2002). Limitations of learning via embeddings in euclidean half-spaces. *JMLR*.

Blum, A. (1992). Learning boolean functions in an infinite attribute space. *Machine Learning*, *9*, 373–386.

Carlson, A., Cumby, C., Rosen, J., & Roth, D. (1999). *The SNoW learning architecture* (Technical Report UIUCDCS-R-99-2101). UIUC Computer Science Dep.

Collins, M., & Duffy, N. (2002). New ranking algorithms for parsing and taggins: Kernels over discrete structures, and the voted perceptron. *Proceedings of ACL 2002*.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines*. Cambridge Press.

Cumby, C., & Roth, D. (2002). Learning with feature description logics. *Proceedings of the 12th International Conference on Inductive Logic Programming*.

Cumby, C., & Roth, D. (2003a). Feature extraction languages for propositionalized relational learning. *IJCAI'03 Workshop on Learning Statistical Models from Relational Data*.

Cumby, C., & Roth, D. (2003b). *Kernel methods for relational learning* (Technical Report UIUCDCS-R-2003-2345). UIUC Computer Science Department.

Freund, Y., & Schapire, R. E. (1998). Large margin classification using the perceptron algorithm. *Computational Learing Theory* (pp. 209–217).

Haussler, D. (1999). *Convolution kernels on discrete structures* (Technical Report UCSC-CRL-99-10). Univerisity of California - Santa Cruz.

Khardon, R., Roth, D., & Servedio, R. (2001). Efficiency versus convergence of boolean kernels for on-line learning algorithms. *NIPS-14*.

Khardon, R., Roth, D., & Valiant, L. G. (1999). Relational learning for NLP using linear threshold elements. *Proc. of the International Joint Conference on Artificial Intelligence* (pp. 911–917).

Kramer, S., Lavrac, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In S. Dzeroski and N. Lavrac (Eds.), *Relational data mining*. Springer Verlag.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*, 285–318.

Novikoff, A. (1963). On convergence proofs for perceptrons. *Proceeding of the Symposium on the Mathematical Theory of Automata* (pp. 615–622).

Roth, D. (1998). Learning to resolve natural language ambiguities: A unified approach. *National Conference on Artificial Intelligence* (pp. 806–813).

Roth, D. (1999). Learning in natural language. *Proc. of the International Joint Conference on Artificial Intelligence* (pp. 898–904).

Roth, D., & Yih, W. (2001). Relational learning via propositional algorithms: An information ext raction case study. *Proc. of the International Joint Conference on Artificial Intelligence* (pp. 1257–1263).

Srinivasan, A., Mugleton, S., King, R. D., & Sternberg, M. (1996). Theories for mutagenicity: a study of first order and feature based induction. *Artificial Intelligence*, *85(1-2)*, 277–299.

Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., & Vapnik, V. (2000). Feature selection for svms. *Neural Information Processing Systems* (pp. 668–674).